VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Webová komponenta pro zobrazování obrazových dat BigDataServeru

Web Based Image Browser for BigDataServer

Petr Kuča

Diplomová práce

Vedoucí práce: Mgr. Ing. Michal Krumnikl, Ph.D.

Ostrava, 2021

Abstrakt

Cílem této práce je navrhnout a implementovat webovou komponentu pro zobrazování obrazových dat poskytovaných BigDataServerem. Obrazová data neboli datasety pochází typicky z *light sheet* techniky mikroskopie a zachycují vývoj biologických organismů. Pro vizualizaci obrazových dat se využívá BigDataViewer plugin do aplikace Fiji ImageJ. Tato práce v prvních částech popisuje principy a technologie BigDataServeru a BigDataVieweru. Následně praktická část pokrývá návrh a implementační detaily webové komponenty, jenž se skládá ze dvou samostatných komponent napsaných v programovacím jazyce Java a React technologii. Závěr této práce je věnován zhodnocení webové komponenty, která umožňuje vizualizaci a základní transformační operace nad obrazovými daty poskytovanými libovolným BigDataServerem.

Klíčová slova

BigDataServer, BigDataViewer, Big Data, vizualizace biologických dat, ImgLib2, afinní transformace, XML/HDF5, TransformJ

Abstract

The goal of this thesis is to propose and implement a web-based component for browsing data provided by BigDataServer. Image data, also named as datasets, comes from the light sheet microscopy technique, which shown the evolution of biological organisms. One possibility of how to visualize data from BigDataServer is to use the BigDataViewer plugin in the Fiji ImageJ application. In the first sections of this thesis, we are describing the principles and technologies of BigDataServer application and BigDataViewer plugin. Description of the practical part contains application design and implementation details. The application consists of two separate components written in Java programing language and React technology. Finally, we evaluate the web-based component which visualizes data from BigDataServer and does basic transformation with image data.

Keywords

BigDataServer, BigDataViewer, Big Data, biological data visualization, ImgLib2, affine transformation, XML/HDF5, TransformJ

Poděkování

Rád bych na tomto místě poděkoval Mgr. Ing. Michalovi Krumniklovi, Ph.D. za cenné rady a připomínky, bez kterých by tato práce nevznikla. Také bych rád poděkoval své přítelkyni Radce, která mi byla při psaní této práce oporou.

Obsah

Seznam použitých zkratek a symbolů

GPL	–	General Public License
JAR	–	Java Archive
HTTP	–	Hypertext Transfer Protocol
XML	–	Extensible Markup Language
JSON	–	JavaScript Object Notation
HDF	–	Hierarchical Data Format
RAM	–	Random Access Memory
URL	–	Uniform Resource Locator
SPIM	–	Selective Plane Illumination Microscopy
HTML	–	Hypertext Markup Language
CSS	–	Cascading Style Sheets
API	–	Application Programming Interface
REST	–	Representational State Transfer
MIT	–	Massachusetts Institute of Technology
JSX	–	JavaScript XML
DOM	–	Document Object Model
JDK	–	Java Development Kit
POJO	–	Plain Old Java Object
ORM	–	Object–relational Mapping
J2EE	–	Java 2 Enterprise Edition
JMS	–	Java Message Service
JMX	–	Java Management Extensions
JDBC	–	Java Database Connectivity
JNDI	–	Java Naming and Directory Interface
TTL	–	Time To Live
UML	–	Unified Modeling Language
OS	–	Operating System
BSD	–	Berkeley Software Distribution

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Existuje mnoho možností, jak zkoumat svět kolem nás. Jednou z nich je technika z oboru mikroskopie zvaná *light sheet*, která umožňuje pozorovat a zkoumat živé biologické vzorky a organismy [1]. Výsledkem této techniky pozorování jsou velmi rozsáhlá 3D časosběrná obrazová data (označována jako datasety), jejichž velikosti šplhají až k terabajtům. Můžeme tedy tyto datasety zařadit do kategorie „Big Data“ (v překladu *velká data*). Následně vzniká potřeba zpřístupnit tyto rozsáhlé datasety pro interaktivní vizualizaci a další analýzu. Avšak na trhu není mnoho *open-source* softwarových řešení, která by nabízela interaktivní prohlížení řezů nebo prohlížení vzorku v čase [2].

Jedním z dostupných softwarů je BigDataViewer plugin aplikace Fiji ImageJ [3], poskytující interaktivní vizualizaci a prohlížení rozsáhlých datasetů. V podstatě se jedná o prohlížeč obrazových dat ve 3D prostoru. V datasetu je uloženo několik sekvencí obrázků, které reprezentují více pohledů na jedny a ty samé data. Tyto pohledy lze v BigDataVieweru barevně rozlišovat, či je skládat dohromady, provádět operace jako rotace, translace, škálování, zoom atd. V neposlední řadě lze také procházet data v určitých bodech v čase (tzv. *timepoints*).

Dataset může být uložen jako soubor na disku nebo může být poskytován skrze BigDataServer. Protože stahování a ukládání rozsáhlých datasetů by bylo vzhledem k jejich běžné velikosti komplikované, nabízí BigDataServer efektivní způsob, jak poskytovat data z takto rozsáhlých datasetů.

Právě BigDataServer a BigDataViewer, respektive jejich technologie jsou hlavními tématy této diplomové práce a tvoří její teoretickou část. Co se týče BigDataServeru, budeme diskutovat mimo jiné následující témata: HDF5 technologii, formát XML/HDF5, instalaci a spuštění BigDataServeru, poskytované webové služby BigDataServerem. V kapitole věnované BigDataVieweru budeme podrobně zkoumat knihovnu ImgLib2 [4], renderovací algoritmus BigDataVieweru a (3D) transformace obrázků pomocí afinních transformací. Afinní transformace si ukážeme na příkladu.

Cílem této diplomové práce je vytvoření webové komponenty pro zobrazování obrazových dat z BigDataServeru, jejíž funkcionalita a ovládání bude vycházet z BigDataVieweru. Pokud by si chtěl totiž uživatel vizualizovat data pomocí BigDataVieweru, musí mít na svém počítači nainstalovanou Fiji ImageJ aplikaci, respektive BigDataViewer plugin a musí znát adresu cílového BigDataSer-

veru. Dále musí uživatel vědět na který *timepoint* se přepnout, případně jak pozorovaný vzorek rotovat, aby dostal smysluplnou vizualizaci. Navrhovaná webová komponenta by měla tento instalační a konfigurační proces zcela eliminovat. Uživatel tedy namísto instalace a zprovozňování Fiji ImageJ aplikace navštíví webovou stránku, která již bude před-konfigurovaná a bude vizualizovat data podobným způsobem jako BigDataViewer. Navíc bude poskytovat funkcionalitu, jenž umožní pozorovaný vzorek dopředu transformovat a vybrat konkrétní *timepoint*, *řez* apod. pro výchozí zobrazení. Popis praktické části bude věnován návrhu, architektuře a implementačním detailům aplikace. Nakonec věnujeme jednu kapitolu zhodnocení vytvořené aplikace a navrhneme, kam by se mohl ubírat její další vývoj.

Jako první si ale v následující kapitole uvedeme a srovnáme další přístupy pro vizualizaci biologických dat, které aktuálně existují. Tato kapitola by měla uvést čtenáře do dané problematiky.

Kapitola 2

Vizualizace biologických dat

Počítačová vizualizace je v biologii široce využívaná ke vhledu do biologických procesů a jejím porozuměním. Metody a nástroje pro vizualizaci biologických dat se za poslední dekádu značně zlepšily, byť pro některé velmi rozsáhlé datové sady jsou stále nedostačující. Může za to hlavně pokrok v oblasti počítačového hardwaru a rozšíření dostupnosti internetu. Neméně důležitým faktorem byl i samotný vývoj široké škály specializovaných metod a nástrojů pro vizualizaci konkrétních druhů biologických dat. Například metody umožňující vizualizaci genomu, makromolekulárních struktur, systémové biologie nebo fylogenetické analýzy [5].

Vizualizační nástroje byly typicky desktopové aplikace navržené pro zobrazování dat pouze z jednoho experimentu. Naproti tomu dnešní nástroje podporují integraci se vzdálenými databázemi a umožňují zobrazovat data z několika zdrojů zároveň. Navíc se stále častěji vytváří nástroje, které umožňují kooperaci s dalšími vizualizačními a analytickými nástroji. Výsledkem takové kooperace několika nástrojů může být například simultánní interaktivní alignment (zarovnání) několika sekvencí s odpovídajícími trojrozměrnými strukturami [6, 7], vizualizace sítě s odpovídajícími tepelnými mapami (*heat maps*), profilovými grafy nebo fylogenetickými stromy a dendrogramy [8]. Mnoho dnešních nástrojů může být přímo umístěno na webové stránky, např. UCSC Genome Browser[1]. Tento nástroj umožňuje zobrazovat sestavené sekvence genomů z různých laboratoří a poskytuje tak přístup k rozmanité škále souvisejících dat.

Vylepšení metod a nástrojů pro vizualizaci biologických dat přineslo ale také exponenciální nárůst velikostí a komplexity zkoumaných biologických dat. Cílem biologů je tedy najít způsob, jak z této záplavy dat vytěžit podstatné informace.

Je dobré si ale uvědomit, že ne vždy je vizualizace vhodným nástrojem. Některé problémy lze lépe řešit výpočetními metodami. Jinde však může vizualizace nabídnout jisté specifické výhody oproti výpočtům. Např. v případě, kdy ještě nejsou v datech známe zákonitosti, poskytuje vizu-

[1] `https://genome.ucsc.edu`

alizace účinný přístup k prozkoumání, a tedy nalezení určitých opakujících se vzorů. Vizualizace může být také užitečná pro projekty, ve kterých doplňuje algoritmické přístupy. V genomice (oboru genetiky) můžou automatizované procesy spolehlivě najít místa, kde dochází k jistému přeskupení, ale vizualizace je pak potřeba pro poskytnutí jakéhosi mentálního obrazu tak, aby byly detaily této strukturální variace více pochopeny.

V této kapitole si dále uvedeme některé další nástroje pro vizualizaci vesměs biologických dat. Nebudeme zabíhat příliš do detailů. Stručně uvedeme o jaký nástroj se jedná a příklady vizualizace tak, aby si mohl čtenář udělat představu.

2.1 Jalview

Jalview je prohlížeč, editor a analytický nástroj pro sekvenční alignment. Jedná se o bioinformatický software napsaný v programovacím jazyce Java. Umožňuje editaci a analýzu rozsáhlých alignmentů (v řádech tisící sekvencí). Případně zobrazení několika sjednocených pohledů (*views*) alignmentů. Podporuje běžné formáty pro sekvence (čtení i zápis) jako FASTA, Clustal, MSF (GCG) nebo PIR.

Jalview je dostupný ve dvou formách. Buď jako samotná desktopová verze Jalview Desktop nebo verze pro webové prohlížeče s názvem JalviewJS. JalviewJS obsahuje také JavaScript API.

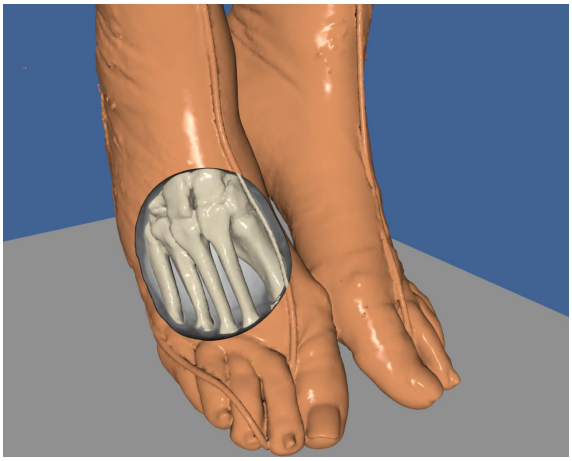
Obrázek 2.1: Jalview - příklad alignmentu z několika různých pohledů (*views*). Zdroj: [9].

Desktopová verze disponuje přístupem k databázi obsahující sekvence proteinů, nukleových kyselin, alignmentů a různých dalších struktur. Obsahuje *Jmol* a *Chimera* prohlížeče pro molekulární struktury nebo program VARNA pro vizualizaci RNA struktur. Kromě grafického uživatelského rozhraní umožňuje také konzervativní analýzu a metody predikce proteinových poruch, poskytované jako *Java Bioinformatics Analysis Web Services* (JABAWS). JABAWS je systém pro spouštění bioinformatických programů, který je možné stáhnout a spustit na libovolném počítači, v clusteru nebo v cloudu [9]. Příklad ukázky alignmentu v aplikaci Jalview můžeme vidět na obrázku 2.1.

2.2 Visualization Toolkit (VTK)

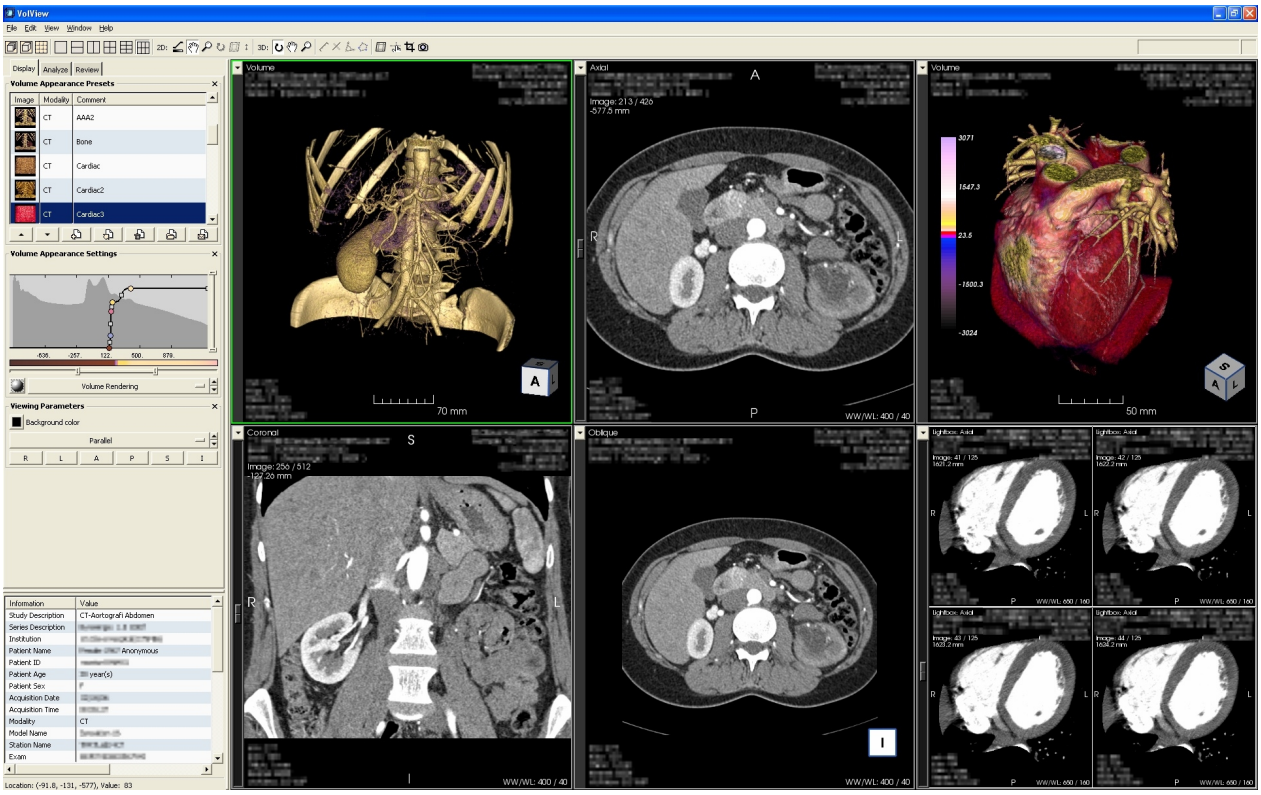
VTK je volně dostupný open-source nástroj pro 3D počítačovou grafiku, modelování, zpracování obrázků, vědecké vizualizace a 2D vykreslování. Podporuje širokou škálu vizualizačních algoritmů jako *scalar*, *vector*, *tensor*, *texture*, *volumetric* metody a pokročilé modelovací techniky jako *implicit modeling*, *polygon reduction*, *mesh smoothing*, *cutting*, *contouring* nebo *Delaunay triangulation*. Vizualizace je realizována přes 3D interaktivní widgety. Je zde podpora pro paralelní zpracování a integraci s dalšími různými GUI nástroji (např. Qt nebo Tk). Jádro aplikace je implementováno v jazyce C++, ale existují i *wrappers* pro programovací jazyky Python, Java a Tcl [10]. Do vývoje software VTK přispívá zejména společnost GE Medical Systems[2] a další výzkumní pracovníci.

Nyní si ukážeme využití aplikace VTK v lékařství. Na obrázku 2.2 můžeme vidět CT[3] scan chodidel s výsekem zobrazující kosti. Na obrázku 2.3 je CT lidského trupu s detailem na ledviny.



Obrázek 2.2: VTK ukázka 1. Zdroj: [11].

[2]Společnost zabývající se inovacemi v oblasti technologií, diagnostiky a digitálních řešení v lékařství.
[3]CT (výpočetní tomografie) je radiologická vyšetřovací metoda.



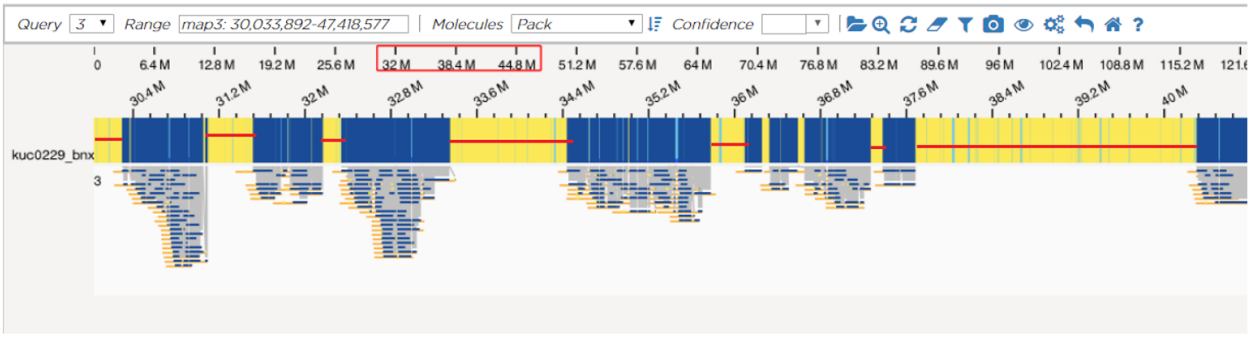
Obrázek 2.3: VTK ukázka 2. Zdroj: [11].

2.3 Bionano Technologie

Společnost Bionano Genomics se zabývá analýzou extrémně dlouhých vysokomolekulární DNA a umožňuje provádět různé výzkumy týkající se genomů. Jedná se o platformu, která zahrnuje přístroj Saphyr pro analýzu vzorků, software pro zpracování a vizualizaci dat (Bionano Access, Bionano Tools). Bionano technologie používá optické mapování anebo de novo sestavení genomu jako metody zkoumání. V principu se ke vzorku DNA přidá speciální látka, která se naváže na její určité části a obarví ji. Obarvená DNA je nasnímána kamerou s vysokým rozlišením a počítačově zpracována. Nasnímaná DNA je pak porovnávaná s referenční DNA a vizualizována (jak ukazuje obrázek 2.4).

Bionano Access je webová aplikace pro mapování genomů od Bionano, správu experimentů, sledování kvality nebo vizualizaci a analýzu dat. Je primárně určená pro data z přístroje Saphyr.

Další software označený jako Bionano Tools, lze stáhnout ze stránek `bionanogenomics.com`. Po rozbalení a instalaci dostaneme nástroje a skripty pro zpracování souborů ve formátu Bionano.



Obrázek 2.4: Příklad vizualizace zarovnání molekul k referenčnímu genomu a výskytu nepokrytých částí (zvýrazněny červenou barvou).

Kapitola 3

Poskytování obrazových dat pomocí BigDataServeru

V této části textu představíme BigDataServer [12], jeho spuštění, parametry pro spuštění a technologie na pozadí.

BigDataServer je minimalistický server využívající protokol HTTP pro zprostředkování dat z datasetů ve formátu XML/HDF5. V souvislosti s BigDataServerem, resp. BigDataViewerem budeme v celé této práci používat termín **dataset**. Tímto termínem budeme zamýšlet množinu 3D obrazových dat neboli svazky obrázků (*image volumes*). Svazky obrázků jsou uspořádaný pomocí ***timepoints*** a ***setups***. V kontextu *light sheet* mikroskopie představuje každý kanál nebo úhel snímaní, případně kombinace obojího, jeden *setup*. Například snímání ve třech úhlech a 2 kanálech představuje celkem 6 *setups*. Každý *setup* v podstatě reprezentuje jeden zdroj dat pro vizualizaci. *Timepoint* představuje jeden konkrétní svazek obrázku. Každá kombinace *setup* a *timepoint* představuje jedno ***view***.

Cílem BigDataServeru je poskytnout data pro vizualizaci v BigDataVieweru. Samotný BigDataServer běží na vzdáleném serveru, kde je uložen jeden nebo více XML/HDF5 datasetů a uživatel si vybírá, který chce zobrazit.

3.1 Instalace a spuštění BigDataServeru

BigDataServer podléhá licenci GPL-3.0. Poslední dostupná verze 3.0.0 byla publikována 20.5.2020. Je distribuován jako jeden JAR soubor, který již obsahuje všechny potřebné závislosti. Pro jeho spuštění je potřeba mít nainstalovanou Javu 7 nebo vyšší. Samotné spuštění provedeme např. příkazem:

```
java -Xmx4G -jar bigdataserver.jar PARAMETRY
```

Poznamenejme, že argument „-Xmx4G“ alokuje 4GB paměti RAM pro spuštění BigDataServeru. Teoreticky může stačit i méně, např. 1GB, ale méně paměti znamená menší prostor pro cache a zhoršuje výkonost aplikace.

3.1.1 Parametry pro spuštění BigDataServeru

BigDataServer má několik parametrů, které lze předat při spuštění, jak ukazuje tabulka 3.1.

Tabulka 3.1: Popis parametrů BigDataServeru

Parametr	Význam	Povinný	Výchozí hodnota
-d <SOUBOR>	Textový soubor specifikující datasety. Na každém řádku je specifikován právě jeden dataset v následujícím formátu: „jméno<tab>XML“.	Ano	
-p <PORT>	Port pro naslouchání příchozích požadavků.	Ne	8080
-s <HOSTNAME>	Síťový název BigDataServeru - specifikuje síťové rozhraní pro naslouchání příchozích požadavků.	Ne	*Hostname serveru*
-t <SLOŽKA>	Složka pro ukládání miniatur. *Pokud není specifikována, vytvoří se nová dočasná složka s názvem „thumbnails“.*	Ne	

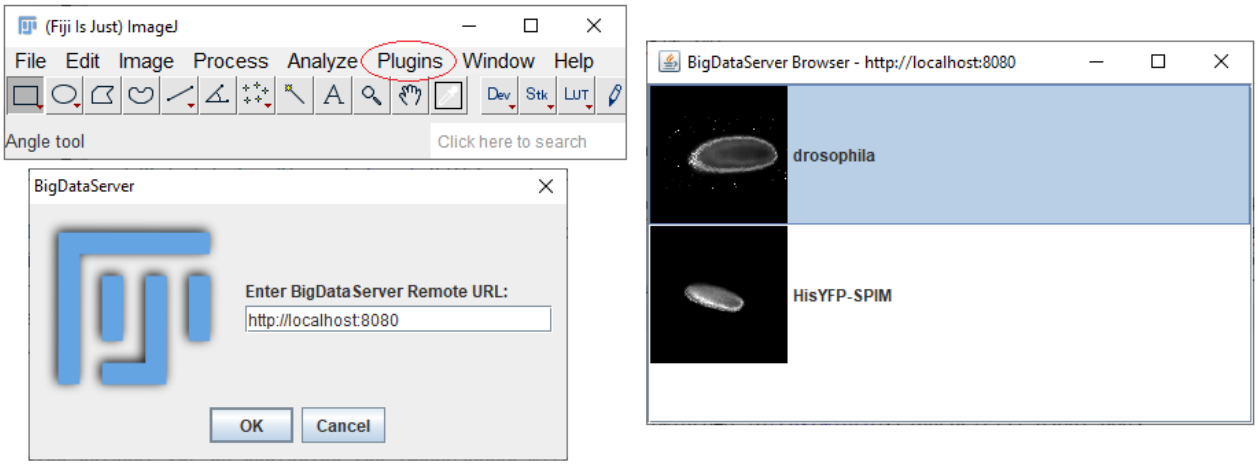
Nyní si ukážeme příklad textového souboru „dataset.txt“, jenž specifikuje datasety BigDataServeru a předává se při spuštění pomocí parametru -d.

```
drosophila drosophila.xml
HisYFP-SPIM HisYFP-SPIM.xml
drosophila_original /mnt/export/hdf5_dataset.xml
```

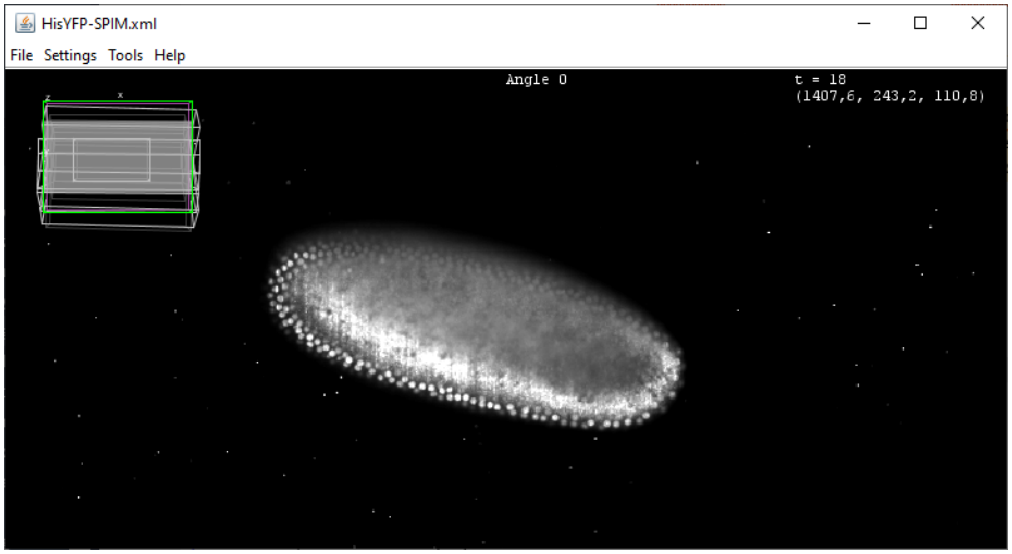
3.2 Připojení k BigDataServeru z Fiji

Pro připojení k BigDataServeru z aplikace Fiji ImageJ zvolíme v horním panelu hlavní okna záložku *Plugins* → *BigDataViewer* → *Browse BigDataServer*. Do dialogového okna vyplníme URL adresu BigDataServeru (včetně portu). Následně se nám zobrazí seznam dostupných datasetů. Dvojklikem myši na vybraný dataset spustíme prohlížení v BigDataVieweru, viz obrázek 3.1.

Prohlížeč BigDataVieweru pak nabízí samostatné nastavení - obrázek 3.2. Stručně si jej shrneme. V záložce *File* je možnost nahrání, resp. uložení aktuálního nastavení pohledu prohlížeče jako „.settings.xml“ soubor. Pod záložkou *Settings* nalezneme: nastavení jasu a kontrastu, výběr aktuálního zdroje (*view*) pro zobrazení, nastavení skupin (skupiny umožňují seskupovat více pohledů a přepínat mezi nimi) nebo možnost zapnout „fused mode“ (zobrazení dat ze všech zdrojů zároveň v překryvu). Záložka *Tools* nabízí nástroje pro práci s obrázky - vytvoření výřezu, nahrání krátkého videa. Nakonec záložka *Help* popisuje navigaci v prohlížeči (více v kapitole 4.3).



Obrázek 3.1: ImageJ Fiji - napojení na BigDataServer. Na obrázku vlevo nahoře můžeme vidět horní panel Fiji aplikace a záložku Plugins přes kterou vybere *Browse BigDataServer*. Na obrázku vlevo dole pak dialog pro zadávaní URL BigDataServeru. Po potvrzení URL se zobrazí seznam dostupných datasetů (obrázek vpravo).



Obrázek 3.2: BigDataViewer - dataset HisYFP-SPIM zachycující embryo octomilky.

3.3 Webové služby poskytované BigDataServerem

BigDataServer implementuje webové API a nabízí několik koncových bodů. V následujících odrážkách si je popíšeme (pořadí odrážek odpovídá jejích volání při připojení z BigDataVieweru):

- `/json/` - V odpovědi nalezneme JSON řetězec popisující dostupné datasety BigDataServeru. Koncový bod nepřijímá žádné parametry. Příklad odpovědi můžeme vidět ve výpisu 3.1.
- `/{datasetName}/png` - Vrací *thumbnail* (miniaturní náhle) pro vybraný dataset ve formátu PNG. Název datasetu je součástí URL cesty.
- `/{datasetName}/?p=init` - Tento koncový bod vrací metadata datasetu v JSON podobě. Jedná se o klíčové data, která nám dávají informace jako počet dostupných *timepoints*, *setups*, *levels*, apod. Ukázku metadat můžeme vidět ve výpisu 3.2. Kromě jména datasetu je nutné předat parametr p s hodnotou „*init*“.
- `/{datasetName}/settings` - Pokud je k dispozici „.settings.xml“ soubor [13] k datasetu, můžeme ho získat pomocí toho koncového bodu. Jedná se o dodatečné nastavení, které umožňuje např. výchozí nastavení jasu. Odpověď je ve formátu XML. Pokud není soubor „.settings.xml“ k dispozici, odpověď bude HTTP kód 404 - *Not Found*.
- `/{datasetName}/?p=cell/{i}/{t}/{s}/{l}/{d1}/{d2}/{d3}/{m1}/{m2}/{m3}` - Koncový bod, jenž vrací samotná obrazová data pro vybraný dataset. Data se vrací v odpovědi jako binární data s HTTP hlavičkou „*Content-Type: application/octet-stream*“. Nutno předat parametr p s hodnotou „cell“ a následující parametry (oddělenými lomítky):
  - `{i}` - *index* hodnota (identifikuje blok dat),
  - `{t}` - *timepoint* hodnota,
  - `{s}` - *setup* hodnota,
  - `{l}` - *level* hodnota,
  - `{d1}` - udává velikost *chunks* (první velikost dimenze),
  - `{d2}` - udává velikost *chunks* (druhá velikost dimenze),
  - `{d3}` - udává velikost *chunks* (třetí velikost dimenze),
  - `{m1}` - specifikuje posun o m_1 *chunks* (první dimenze),
  - `{m2}` - specifikuje posun o m_2 *chunks* (druhá dimenze),
  - `{m3}` - specifikuje posun o m_3 *chunks* (třetí dimenze).

  Např. URL `/datasetPetrPipeline02/?p=cell/100/0/0/2/16/16/16/64/0/48` vrátí data pro dataset „datasetPetrPipeline02“ dostupná pod *timepoint* 0, *setup* 0 a *level* 2. Velikost

bloků (*chunks*) je v tomto případě 16x16x16 a je vracen blok dat posunut o 64 jednotek v první dimenzi, 0 jednotek v druhé dimenzi a 48 jednotek v třetí dimenzi. Navíc jsou tato data označena indexem 100 a uložena pod tímto indexem do cache BigDataServeru.

```
{   "datasetSona": {
      "id": "datasetSona", "description": "Dataset Sona",
      "qcmpSupported": true,
      "thumbnailUrl": "http://julius2.it4i.cz:80/datasetSona/png",
      "datasetUrl": "http://julius2.it4i.cz:80/datasetSona/" },
    "datasetPetrPipeline02": {
      "id": "datasetPetrPipeline02", "description": "Dataset Petr - Pipeline 02",
      "qcmpSupported": true,
      "thumbnailUrl": "http://julius2.it4i.cz:80/datasetPetrPipeline02/png",
      "datasetUrl": "http://julius2.it4i.cz:80/datasetPetrPipeline02/"} }
```

Výpis 3.1: Dostupné datasety BigDataServeru `julius2.it4i.cz`.

```
{  "dimsAndExistence": [
        [  { "level": 0, "setupId": 0, "timepointId": 4 },
           { "dimensions": [ 1045, 996, 945 ], "exists": true } ],
        [...]
    ],
    "maxNumLevels": 4, "maxNumSetups": 2, "maxNumTimepoints": 140,
    "perSetupMipmapInfo": {
        "0": { "maxLevel": 3, "resolutions": [...], "subdivisions": [...],
               "transforms": [...] },
        "1": {...} } }
```

Výpis 3.2: JSON metadata datasetu „datasetPetrPipeline02“ (zkrácený výpis). Názvy většiny atributů jsou samopopisné. Zmíníme jen atributy „resolutions“, „subdivisions“ a „transforms“. Tyto atributy obsahují data související s *multi-resolution pyramids*.

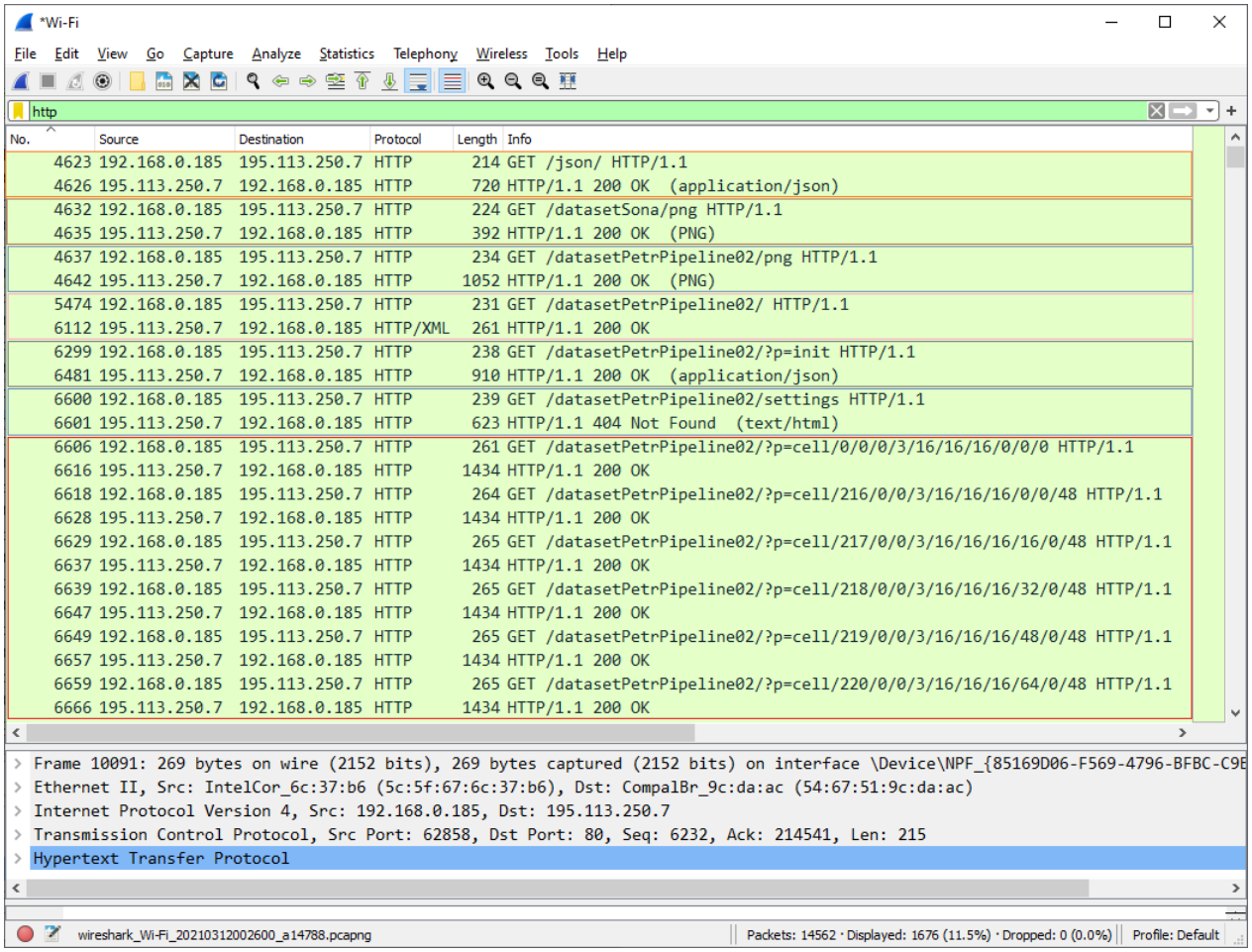
3.3.1 Analýza síťové komunikace

V této podkapitole se podíváme blíže na síťovou komunikaci mezi BigDataServerem a BigDataViewerem. Využijeme k tomu program Wireshark, který slouží k analyzování síťového provozu a můžeme si v něm prohlížet zachycené pakety.

Byla zachytávána síťová komunikace, kde cílový BigDataServer byl veřejně přístupný pod URL adresou `julius2.it4i.cz` (IP adresa 195.113.250.7) a my jsme na něj byli napojeni z lokálního

počítače (IP adresa 192.168.0.185). Část komunikace zachycuje obrázek 3.3 přímo v aplikaci Wireshark. Barevně jsou zvýrazněny HTTP požadavky na API BigDataServeru, viz předchozí kapitola. Na ukázce můžeme vidět jako první požadavek s URL adresou `/json/` a PNG miniatury (*thumbnails*) jako odpověď. Data z těchto požadavků jsou využita pro zobrazení dostupných datasetů v BigDataVieweru.

Po výběru datasetu, v tomto případě „datasetPetrPipelin02“, se inicializuje stažení metadat a dodatečného nastavení. Následně začne stahování obrazových dat.



Obrázek 3.3: Analýza síťové komunikace pomocí aplikace Wireshark

3.4 HDF5 technologie

HDF5 technologie [14, 15] slouží pro efektivní ukládání a správu rozsáhlých, rozmanitých dat. Tvoří ji seskupení data modelu, softwarů a souborového formátu. Současně s daty jsou také ukládána jejich metadata. Technologie podporuje ukládání velké škály datových typů[1]. Sada technologií HDF5 zahrnuje také nástroje pro správu, manipulování, prohlížení a analýzu HDF5 dat. Jedním z takových nástrojů pro manipulaci s HDF5 daty je **BigDataServer**, i když ten využívá rozšířený XML/HDF5 formát o kterém si řekneme více v kapitole 3.5. **BigDataViewer** patří zase mezi nástroje pro prohlížení HDF5 dat. V tomto úvodu o HDF5 technologii ještě zmíníme její přenositelnost a rozšiřitelnost. Tyto vlastnosti umožňují aplikacím, které používají HDF5 další vývoj [16]. Samotnou technologii vyvíjí korporace „The HDF Group“ - `https://www.hdfgroup.org`.

3.4.1 Data Model

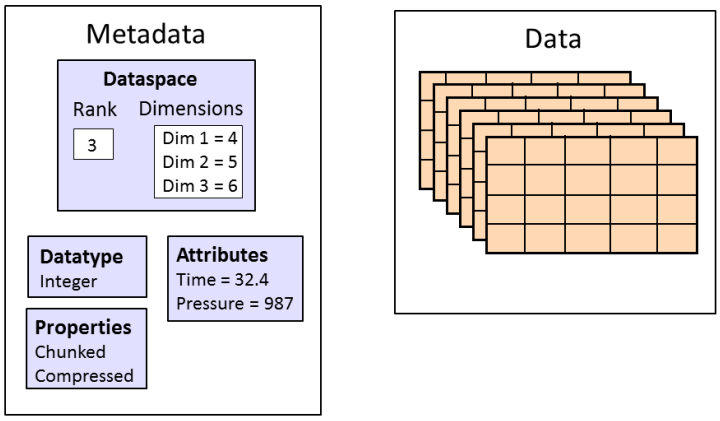
HDF5 data model, také nazývaný jako abstraktní nebo logický model, definuje tzv. bloky pro organizaci dat. Dle [16] data model definuje HDF5 *information sets* nebo zkráceně *infoset*. *Infoset* je v podstatě kontejner pro anotované sdružení polí, skupin a datových typů. Tyto sdružená pole, skupiny a datové typy se postupně v data modelu nazývají HDF5 datasety, HDF5 skupiny, respektive HDF5 datové typy. HDF5 data model definuje také buďto jednoduché, nebo pokročilé mechanismy pro vytváření vazeb (sdružovaní) HDF5 datasetů.

HDF5 dataset

HDF5 datasety jsou proměnné typu pole, jejichž datové prvky jsou logicky rozloženy a uloženy jako vícerozměrná pole. Tyto vícerozměrná pole obsahují jak data samotná, tak metadata, která popisují data samotná. Lépe nám to ukáže obrázek 3.4.
Velikost HDF5 datasetu může růst v souladu s jeho limity. V závislosti na vybrané strategii úložiště (*storage layout strategy*) může být maximální velikost omezená, nebo neomezená. Současně podporované strategie jsou:

- *Contiguous*. Prvky pole jsou rozloženy do jedné sekvence v HDF5 polích.
- *Compact*. Rozložení prvků do menších HDF5 datasetů (menších než 64KB).
- *Chunked*. Prvky pole jsou rozloženy jako kolekce pravidelných dílčích polí o pevné velikosti neboli bloků nazývaných *chunks*[2]. (Neomezená maximální velikost růstu datasetu může být nastavena buď pro všechny, nebo jen některé dimenze.)

[1] Výčet datových typů je dostupný na `https://support.hdfgroup.org/HDF5/doc1.6/UG/11_Datatypes.html`
[2] *Chunks* by se daly volně přeložit jako kusy dat.



Obrázek 3.4: Znázornění HDF5 datasetu. Zdroj: [14]. Na obrázku je zobrazen příklad datasetu, který ukládá data jako tří dimenzionální dataset o rozměrech dimenzí 4 x 5 x 6 a jednotlivé prvky jsou datového typu *integer*. Parametry datasetu, jenž popisuje tabulka *Attributes*, jsou čas (time) a tlak (pressure). Tabulka *Properties* říká, že dataset je typu chunked (rozdělen do bloků) a je použita komprese. Hodnota *Rank* zachycuje počet dimenzí, v tomto případě je tři.

Vybraná strategie má dopad na rychlost operací nad HDF5 datasetem. Např. *contiguous* strategie má téměř konstantní dobu přístupu ke kterémukoliv elementu z pole a nulovou režii. Vstupní obrazová data pro BigDataServer v současné době disponují strategií *chunked*.

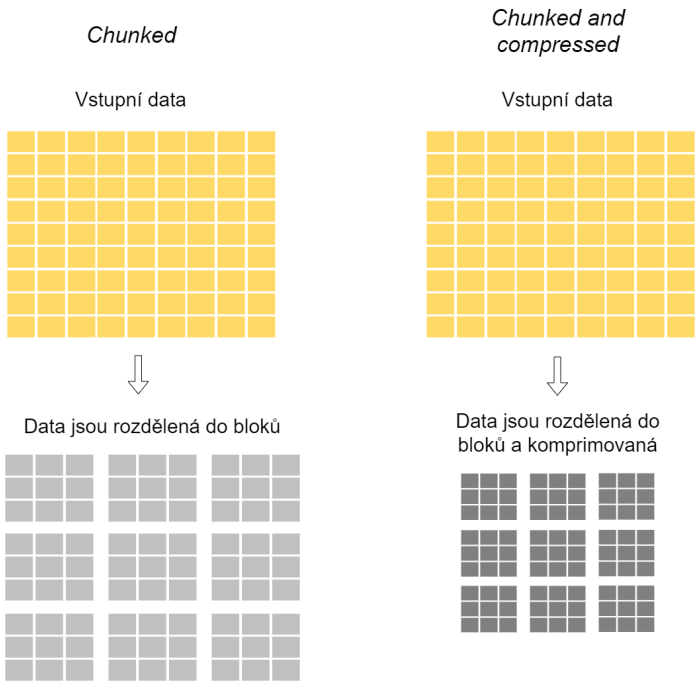
Chunked HDF5 datasety nabízí flexibilitu a můžou podstatně zlepšit čas přístupu na určitou část pole (řez). Příklad takového datasetu jsme mohli vidět na obrázku 3.4. Rychlejšího přístupu můžeme také dosáhnout, pokud je tato strategie použita současně s kompresí (*chunked and compressed*) - znázorňuje obrázek 3.5. V případě užití komprese se nekomprimují pouze referenční *chunks* [16].

Chunks pro datasety BigDataServeru jsou tří dimenzionální s typickými velikostmi 32x32x32 nebo 16x16x16 a můžou růst neomezeně ve všech dimenzích. Jako komprimační metoda se často používá GZIP, úroveň komprese 6. Obsahem HDF5 datasetu můžou být tabulky, grafy, dokumenty (např. PDF) nebo jako v případě BigDataServeru obrázky.

HDF5 skupiny

HDF5 skupiny organizují data objekty. Každý HDF5 soubor má kořenovou skupiny, ta může obsahovat více vnořených skupin. Asociace skupin je provedena pomocí tzv. *links*. Práce s HDF5 skupinami je podobná práci se soubory a složkami v UNIX operačních systémech. Objekty jsou často reprezentovány (absolutní) cestou. Např.:

- / - značí kořenovou (root) skupinu,
- /t00000 - značí člena kořenové skupiny pojmenovaného jako „t00000“,



Obrázek 3.5: Chunked a Chunked and Compressed strategie

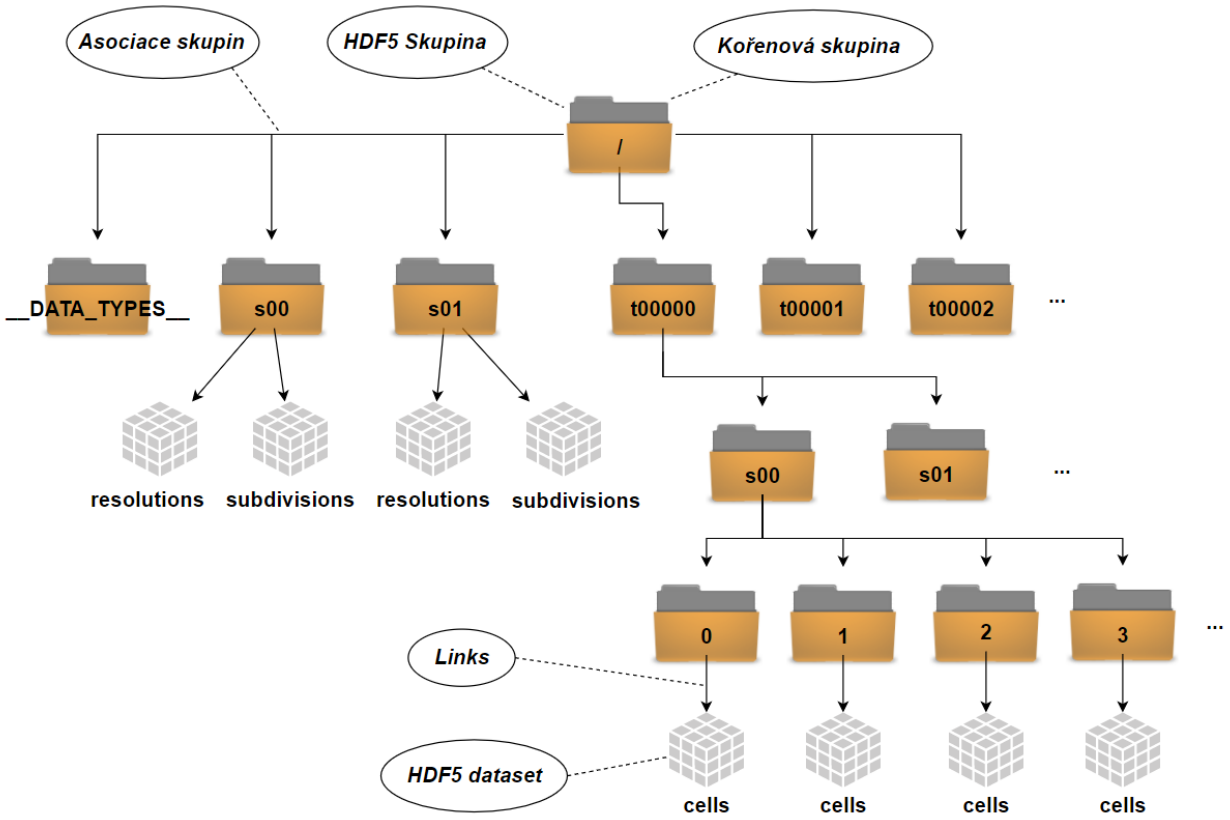
- /t00000/s00 - značí skupinu „s00“, která je členem skupiny „t00000“ a ta je členem kořenové skupiny.

Obrázek 3.6 ukazuje HDF5 skupiny, HDF5 datasety a jejich asociace na reálném příkladu datasetu pro BigDataViewer.

HDF5 datové typy

Datové typy popisují konkrétní prvky dat v HDF5 datasetu. Poskytují též kompletní informace o konverzi dat z, nebo do konkrétního datového typu. Dle [17] můžeme uvést následující podporované typy proměnných: *integer* (celočíselné), *floating-point* (s pohyblivou řádovou čárkou), *character* (znaky), *string* (řetězce), *date and time* (datum a čas), *bitfield* (bitová pole), *enum* (výčtové typy), *array* (pole), *opaque*[3], *compound*[4], *reference*[5] a *variable-length*[6]. Většina z nich je všeobecně známa, a tedy i jejich použití a hodnoty si lze představit. U specifických proměnných (*opaque*, *compound*, *reference*, *variable-length*) je potřeba si nastudovat dokumentaci [17], aby nedošlo k nesprávnému nebo neefektivnímu použití.

[3]Neinterpretovaná data - posloupnost bajtů, která se ukládá a načítá jako blok.
[4]Datový typ složený z více datových typů.
[5]Reference na jiný objekt v HDF5 souboru.
[6]1-dimenzionální pole s proměnou délkou.



Obrázek 3.6: Ukázka HDF5 skupin na reálném datasetu. Obrazová data jsou uložena jako HDF5 datasety (na obrázku jsou označovány jako *cells*) a jsou organizovány do složek podle *timepoints* (složky t00000, t00001, t00002, atd.), *setups* (složky s00, s01) a *levels* (složky 0, 1, 2, 3). Nakonec složky s00 a s01 spojené s kořenovou skupinou obsahují *resolutions* a *subdivisions* datasety. Tyto datasety obsahují data týkající se mipmap pyramid.

Datové typy v HDF5 můžeme také rozdělit na tyto dvě skupiny:

1. **Předdefinované datové typy** (*pre-defined datatypes*): Tyto datové typy definuje přímo HDF5 technologie. V průběhu vývoje HDF5 se můžou měnit. Existují dva typy předdefinovaných datových typů:

   (a) **Standardní datové typy** - mají stejné hodnoty a chování na všech platformách. Jejich pojmenování má tvar H5T_ARCH_BASE, kde ARCH je název architektury a BASE je název typu. Např. H5T_IEEE_F32BE značí standardní datový typ s pohyblivou řádovou čárkou pro Big Endian.

   (b) **Nativní datové typy** - nativní datové typy jsou primárně určeny pro zjednodušení práce s pamětí (čtení, zápis). Nevýhoda je, že nejsou na všech platformách stejné. Např. H5T_NATIVE_INT představuje nativní proměnou typu *int* (programovacího jazyka C).

2. **Odvozené datové typy** (*derived datatypes*): jsou vytvořeny nebo odvozeny z předdefinovaných datových typů. Například datový typ *string* představuje jeden nebo více *character*, je tedy odvozený. Do odvozených datových typů patří také již zmíněny *compound*.

Pokud se nyní podíváme opět na reálný dataset pro BigDataViewer zjistíme, že ten používá 16-bit *integer* jako datový typ pro obrazová data.

3.4.2 Software

Software pro HDF5 je napsaný v programovacím jazyce C. Distribuci HDF5 lze stáhnout na jejich domovské stránce `www.hdfgroup.org` a zahrnuje knihovny, nástroje pro podporu práce z příkazové řádky, potřebné soubory a skripty pro kompilaci, i několik ukázkových programů. Software obsahuje také tzv. *wrappers* pro ostatní programovací jazyky jako C++, FORTRAN (90 a F2003) a Java.

Software dále zahrnuje také HDF5 API a knihovny pro práci se třemi důležitými objekty HDF5. (Všechny rutiny programovacího jazyka C začínají prefixem „H5“ a následuje písmeno označující typ objektu.) Tedy: H5**A** - rozhraní pro prací s atributy HDF5 datasetu (**A**ttributes), H5**D** - rozhraní pro prací s HDF5 datasetem samotným (**D**atasets), H5**F** - rozhraní pro práci s HDF5 soubory (**F**iles).

Mezi nástroje užitečné pro práci s HDF5 soubory můžeme zařadit: *h5dump* - nástroj pro výpis nebo zobrazení obsahu souboru HDF5; *h5cc, h5c++, h5fc* - skripty pro kompilaci aplikací v UNIX OS; HDFView - desktopová Java aplikace pro prohlížení souborů (podporuje formáty HDF4 a HDF5).

3.4.3 Souborový formát

Souborový formát nebudeme nějak detailně rozepisovat. Zmíníme jen, že formát souboru HDF5 je definován jeho specifikací - HDF5 File Format Specification [18]. Ta určuje fyzickou organizaci

bitů a jejich uložení na paměťovém médiu. Soubory můžou mít následující přípony: „.hdf“, „.h5“, „.hdf5“, „.he5“.

3.5 XML/HDF5 formát

BigDataServer, resp. BigDataViewer, využívá vlastní XML/HDF5 formát. Jedná se o speciální hierarchický datový formát, který s pomocí ImgLib2 knihovny optimalizuje přístup k jakékoliv části velkých datasetů. Knihovnu ImgLib2 budeme diskutovat v kapitolách 4.4 a 7.7.1. Zjednodušeně řečeno, dataset ve formátu XML/HDF5 se ze vstupního obrázku vytváří následujícím způsobem:

- Pokud obrázek obsahuje více kanálu (*channels*), každý kanál se stane jedním **setup**.
- Pokud obrázek obsahuje sekvenci dalších obrázků (tzv. *frames*), každý takový obrázek z dané sekvence se stane **timepoint**.

Budeme-li chtít vytvořit dataset ve formátu XML/HDF5, můžeme k tomu využít Fiji ImageJ aplikaci, konkrétně BigDataViewer plugin a položku *Export Current Image as XML/HDF5* [13]. Výstupem bude XML soubor, který obsahuje metadata (popisující daný dataset) a „.h5“ soubor. Tyto dva výstupní soubory můžeme použít jako vstupní dataset pro BigDataServer. Soubor typu „.h5“ neboli HDF5 technologii jsme si popsali v předchozí sekci. V následující podkapitole si popíšeme XML soubor s metadaty a jeho strukturu.

3.5.1 XML soubor

V této sekci budeme pracovat s výpisem 3.3. Ve výpisu můžeme vidět část XML souboru s metadaty jako ukázku. Jedná se o metadata k minimalistickému testovacímu datasetu zachycující embryo cctomilky[7]. Dataset obsahuje 2 *setups* a 3 *timepoints*, celkem tedy 6 *views*.

Kořenovým elementem XML souboru je <SpimData>. Kořenový element musí dále obsahovat minimálně <BasePath>, <SequenceDescription> a <ViewRegistrations> elementy (v libovolném pořadí).

<BasePath> element (řádek 3) definuje výchozí cestu pro všechny relativní cesty v souboru. Často se zadává jako ‘.’, stejně jak je tomu v ukázce, což značí složku, kde se nachází samotný XML soubor.

<SequenceDescription> element (řádek 4-22) definuje *setups*, *timepoints* a *views* obsažené v datasetu. Jako první můžeme v ukázce vidět <ImageLoader> element (řádek 5-7), který popisuje zdroj obrazových dat pro každé *view*. Jako atribut <ImageLoader> elementu specifikujeme formát zdroje. V našem případě je to „bdv.hdf5“, což znamená, že obrazová data se budou číst z HDF5 souboru. Obsah <ImageLoader> elementu je pak závislý na specifikovaném formátu.

[7] Dataset lze stáhnout na `http://fly.mpi-cbg.de/~pietzsch/bdv-examples/` (drosophila.xml, drosophila.h5).

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <SpimData version="0.2">
3    <BasePath type="relative">.</BasePath>
4    <SequenceDescription>
5         <ImageLoader format="bdv.hdf5">
6             <hdf5 type="relative">drosophila.h5</hdf5>
7         </ImageLoader>
8         <ViewSetups>
9             <ViewSetup>
10                <id>0</id>
11                <name>angle 1</name>
12            </ViewSetup>
13            <ViewSetup>
14                <id>1</id>
15                <name>angle 2</name>
16            </ViewSetup>
17      </ViewSetups>
18        <Timepoints type="range">
19            <first>0</first>
20            <last>2</last>
21        </Timepoints>
22    </SequenceDescription>
23    <ViewRegistrations>
24        <ViewRegistration timepoint="0" setup="0">
25            <ViewTransform type="affine">
26                <affine>0.996747 -0.00182 -0.01594 -3.92997 -0.00590 0.492163 -3.01238 262.5342
                      0.004747 0.868287 1.771636 -426.667</affine>
27            </ViewTransform>
28        </ViewRegistration>
29        <ViewRegistration timepoint="0" setup="1">
30        [...]
31    </ViewRegistrations>
32 </SpimData>
```

Výpis 3.3: Metadata datasetu ve formátu XML - příklad (zkrácený výpis)

Pro HDF5 formát stačí jednoduše specifikovat jméno souboru (řádek 6). Dalším elementem, který můžeme nalézt v <SequenceDescription> je <ViewSetups> (řádek 8-17).

<ViewSetups> obaluje sekvenci <ViewSetup> elementů. Jak název elementu napovídá, jedná se o popis konkrétních *views*. Každý <ViewSetup> musí mít unikátní <id> (0, 1,...) a může specifikovat další libovolné atributy jako např. <voxelSize>, <size>, apod. Více informací o těchto volitelných atributech můžeme nalézt ve zdroji [13] uvnitř sekce 2.5. V ukázce můžeme nalézt pojmenování *view* jako volitelný atribut (element <name> na řádku 11 a 15).
<Timepoints> element (řádek 18-21) popisuje dostupné *timepoints* a může být zadán několika způsoby: jako rozsah (range), jako seznam (list) nebo jako vzor (pattern). V naší ukázce jsou *timepoints*

zadány jako rozsah počínaje *timepoint 0* a *timepoint 2* konče.

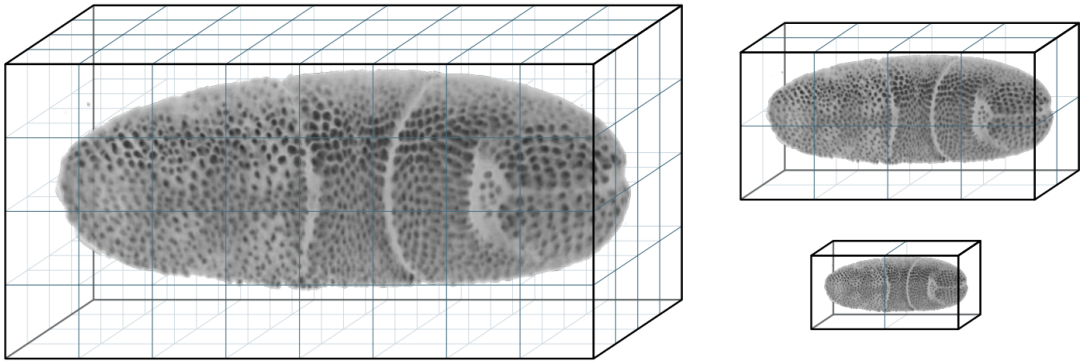
Posledním elementem z ukázky, který nám zbývá popsat, je <ViewRegistrations> (řádek 23-66). Ten popisuje transformaci pro každé *view*, pomocí které transformujeme *view* do globálního souřadnicového systému. O souřadnicových systémech bude kapitola 4.4.5. Co si představit pod transformací do globálního souřadnicového systému se dozvíme v kapitole 4.5. Jak již bylo zmíněno, v naší ukázce můžeme vidět 6 *views*: (timepoint 0, setup 0), (timepoint 0, setup 1) atd. až po (timepoint 2, setup 1). Každému z těchto 6 *views* náleží právě jeden <ViewRegistration> element, jenž specifikuje 3D afinní matici. Matice může být zadaná buď pomocí jednoho <ViewTransform> elementu (řádek 25-28) nebo jako seznam <ViewTransform> elementů a jejich spojením dostaneme finální transformační matici [13]. Všimněte si atributu *type* a hodnoty „affine“ u <ViewTransform> elementu, který značí, že se jedná o afinní transformační matici. Transformační matice v tomto případě představuje vstup pro tzv. inicializační transformační operace obrazových dat, viz kapitoly 4.7 a 4.8.

3.6 Mipmap Pyramidy

Obrazová data jsou v HDF5 souboru uložena jako tzv. *multi-resolution* pyramidy. To znamená, že kromě původního (vysokého) rozlišení se uloží také několikrát zmenšené původní rozlišení jako tzv. *mipmaps*. To ve výsledku slouží ke dvěma účelům. Ten první je minimalizace aliasing efektu při renderování oddáleného *view* datasetu [19]. Druhým a hlavním důvodem je to, že použitím *mipmaps* dojde k výraznému zkrácení doby přístupu k datům a zvýšení odezvy aplikace při navigaci. Důvody jsou zřejmé, menší rozlišení vyžaduje méně paměti a data jsou z disku nahrávána rychleji. Při prohlížení datasetu se musí vždy nové bloky dat (*chunks*) nahrát do paměti. (Pokud již nejsou uložené v cache.) V takové situaci může BigDataViewer podstatně rychleji načíst a vykreslit data s nižším rozlišením. Vykreslení detailů z vyššího rozlišení je uskutečněno později, jakmile jsou data s vyšším rozlišením stažena. Proces vykreslování detailů je spuštěn, jakmile uživatel přestane „hýbat“ s prohlížečem BigDataVieweru. Pojďme se nyní podívat na obrázek 3.7, který nám znázorňuje *multi-resolution* pyramidy na 3D obrázku.

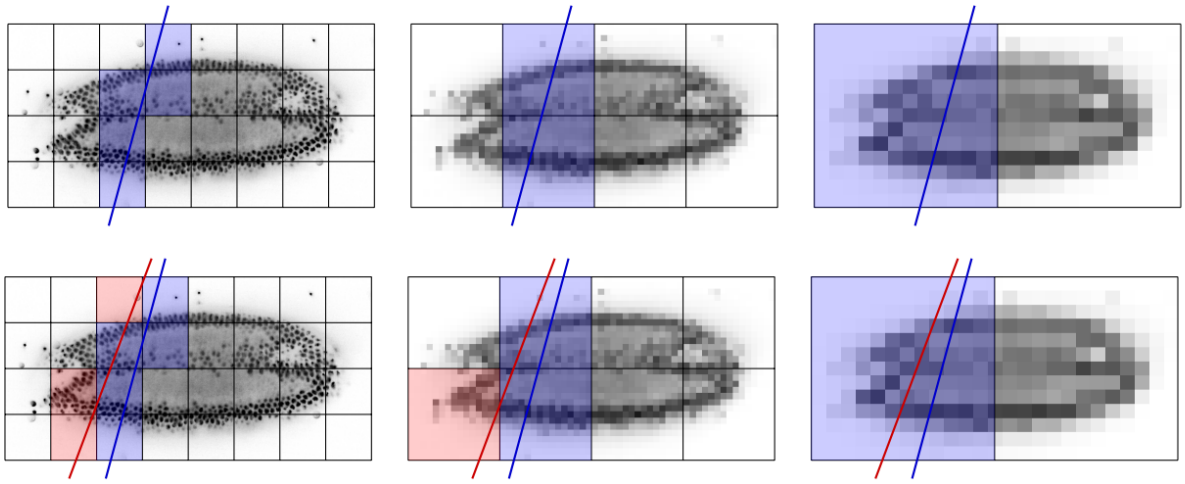
Každá úroveň (*level*) multi-resolution pyramidy je uložena jako bloky (*chunks*) multidimenzionálního pole. Multidimenzionální pole jsou klasický způsob, jakým jsou data uložena v HDF5 souboru, viz kapitola 3.4. S použitím *multi-resolution* pyramid je pak při renderování potřeba jen část dat z bloků. Nicméně, každý z těchto bloků se načítá do paměti kompletně, i když je potřeba jen část dat. Načítání dat tímto způsobem, tedy zarovnaných na hranici bloků, zaručuje optimální výkon I/O (vstupně-výstupních) operací.

Všechny takto nahrané bloky jsou uloženy v paměti RAM a představují cache data. Protože se typicky v průběhu prohlížení řezů mění pozice pozorovaného vzorku jen mírně, podobné množiny řezů (*chunks*) jsou protnuty. Tj. využívají se cache data uložena v RAM paměti. Z disku je třeba načítat pouze ty bloky, které aktuálně nejsou v cache. V kombinaci s reprezentací *mipmap* to umož-



Obrázek 3.7: Multi-resolution pyramidy. Zdroj: [13]. Každý svazek obrazových dat je uložen ve více rozlišení. Původní rozlišení (vlevo) a následně jeho menší verze (vpravo). Každé takové rozlišení je uloženo jako více bloků dat rozdělených do menších 3D bloků.

ňuje plynulé a interaktivní procházení velmi velkých datových sad [13]. Tuto strategii vizualizuje obrázek 3.8.



Obrázek 3.8: Nahrávání a cache bloků dat (*chunks*). Upraveno dle [13]. Modrá přímka znázorňuje renderovaný řez. V původním rozlišení je potřeba načíst 5 bloků dat. Kdežto pouze 2, resp. 1 ve zmenšeném rozlišení. Detaily z vysokého rozlišení jsou vyplněny posléze. Pokud následně změníme prohlížený řez (obrázky ve druhém řádku), můžou se použít již data uložená v paměti - modrá přímka. A pouze část chybějících bloků se načítá (jsou označeny červenou přímkou).

Kapitola 4

Vizualizace dat pomocí komponenty BigDataViewer

V této kapitole si popíšeme BigDataViewer a lehce si znázorníme princip *light sheet mikroskopie*. Následně si popíšeme ImageJ program a některé jeho distribuce. Větší část textu budeme věnovat knihovně ImgLib2 a jejím funkcím, které BigDataViewer využívá (např. virtualizovaná interpolace, virtualizované rozšíření obrázku apod.) Podrobně se pak budeme věnovat transformacím obrázku a afinním transformacím, protože ty pak budou využity v implementaci.

4.1 BigDataViewer

BigDataViewer je prohlížeč pro sekvence obrázků o velikosti v rámci terabajtů. Umožňuje interaktivní navigaci a vizualizaci dat získaných pomocí *light sheet* mikroskopie. Jedná se o metodu fluorescenční mikroskopie, při které je pozorovaný vzorek osvětlován pomocí excitačního paprsku vytvarovaného do plochy. Paprsek prochází pozorovaným vzorkem kolmo k objektivu a vzorek je tak možné postupně nasnímat ve více rovinách. Následně je možné jej analyzovat, provádět 3D rekonstrukce a vytvářet časosběrné fotografie [1]. Základní princip a sestavení *light sheet* mikroskopie můžeme vidět na obrázku 4.1.

BigDataViewer je součástí Fiji distribuce jako ImageJ plugin. Lze ho najít v horním menu pod záložkou *Plugins* → *BigDataViewer*. V následujících kapitolách stručně popíšeme, co je to ImageJ a jaké jsou jeho nejznámější distribuce; použití ImgLib2 knihovny pro renderování a základní renderovací algoritmus na kterém je BigDataViewer postaven.

4.2 ImageJ

ImageJ je univerzální open-source software pro zpracování a analýzu obrázků především z oblasti vědy. Software je napsaný v programovacím jazyce Java. Existuje několik distribucí založených na

Obrázek 4.1: Princip light sheet mikroskopie. Zdroj světla se nachází kolmo k optické dráze světla. Snímaný vzorek je osvětlován pomocí úzké roviny světla, tzv. *light sheet*. Upraveno dle [20].

ImageJ, kde samotný ImageJ tvoří základ, který obsahuje základní nástroje pro načítání, zobrazování, zpracování a export obrázků. Ostatní distribuce přidávají k tomuto základu další funkcionalitu navíc určenou pro specifické obory vědy.
Jedna z hlavních distribucí používaných pro obory biologie je **Fiji** [3] (**F**iji **I**s **J**ust **I**mageJ). Distribuci lze najít na webové adrese `http://fiji.sc/Fiji`. Fiji nabízí velkou škálu předinstalovaných maker a pluginů (mimo jiné také BigDataViewer), které se zaměřují na zpracování dat z fluorescenčních mikroskopů. Obsahuje například nástroje a pluginy pro segmentace obrazu, registrace obrazu, kolokalizace a další. Podporuje velké množství formátů pro ukládání obrázků. Vedle standardních formátu jako JPEG, GIF, BMP, PNG, také tzv. *Bio-Formats*. Fiji obsahuje také vlastní uživatelsky přívětivý skript editor.

Pro oblasti astronomie existuje **AstroImageJ** distribuce dostupná na adrese `http://www.astro.louisville.edu/software/astroimagej`. AstroImageJ mírně modifikuje základní distribuci ImageJ, takže s ní není přímo zpětně kompatibilní. Distribuce doplňuje speciální makra a pluginy pro potřeby analýzy v astronomii.

Poslední distribucí, kterou zmíníme je **μManager**. Webová stránka tohoto projektu je `https://www.micro-manager.org`. μManager nabízí rozhraní zaměřené na získávání obrázků a ovládání hardware. Z jeho pomocí můžeme přímo nahrávat data „online“ z kamery nebo mikroskopu do ImageJ pro jeho další zpracování. Příkladem může být projekt Open SPIM, kde se μManager používá pro ovládání světla mikroskopu, získávání obrázků a jejich zpracování [21].

4.3 Základní navigace a ovládání aplikace

Po připojení k BigDataServeru z Fiji ImageJ a vybrání konkrétního datasetu, viz kapitola 3.2, se zobrazí BigDataViewer prohlížeč. Jakmile jsou data načtena, dojde k zobrazení prostředního řezu

prvního dostupného zdroje dat (úhlu). BigDataViewer můžeme ovládat pomocí myši a klávesnice.

Úplně základní možnost je prohlížení řezů. Při prohlížení řezů se pohybujeme na ose Z. Řezy si prohlížíme buďto za použití kolečka myši, nebo pomocí kláves $\boxed{<}$ (oddálení), $\boxed{>}$ (přiblížení) na klávesnici. Translaci neboli pohybování pozorovaným vzorkem na osách X a Y můžeme provést stisknutím $\boxed{\text{pravého tlačítka myší}}$ nebo $\boxed{\text{kolečka myši}}$ a následným pohybem kurzoru po plátně. Pro přibližování (zoom in), respektive oddálení (zoom out) použijeme buďto kombinaci $\boxed{\text{Shift}} + \boxed{\text{Ctrl}} + \boxed{\text{kolečko myši}}$ nebo šipky $\boxed{\uparrow}$ (zoom in), $\boxed{\downarrow}$ (zoom out). Poslední navigace, která patří mezí základní je rotace. Rotaci můžeme provést stisknutím $\boxed{\text{levého tlačítka myší}}$ a táhnutím myši po plátně ve směru, kterým chceme rotovat.

V tabulce 4.1 můžeme nalézt souhrn klávesových zkratek pro ovládání BigDataVieweru. Lze také využít možnosti rychlého a pomalého prohlížení. Např. pokud budeme při rotaci držet klávesu $\boxed{\text{Shift}}$, bude rotace 10x pomalejší. Případně pří stisknutí klávesy $\boxed{\text{Ctrl}}$ bude rotace 10x rychlejší. Poznamenejme, že při otáčení okolo os (například při stisknutí kláves $\boxed{\text{Shift}} + \boxed{\text{X}}$) se rotuje okolo aktuální pozice kurzoru myši.

Tabulka 4.1: Navigace BigDataVieweru pomocí klávesových zkratek

$\boxed{\text{X}}$, $\boxed{\text{Y}}$, $\boxed{\text{Z}}$	Výběr osy otáčení (X, Y, Z).
$\boxed{\cdot}$, $\boxed{,}$	Oddálení, resp. přiblížení na ose Z.
$\boxed{\leftarrow}$, $\boxed{\rightarrow}$	Rotace ve směru, resp. proti směru hodinových ručiček okolo zvolené osy.
$\boxed{\text{Shift}} + \boxed{\text{X}}$	Otáčení na rovině ZY. Jinými slovy také pohled podél osy X aktuálně vybraného zdroje (úhlu).
$\boxed{\text{Shift}} + \boxed{\text{Y}}$	Otáčení na rovině XZ. Jinými slovy také pohled podél osy Y aktuálně vybraného zdroje (úhlu).
$\boxed{\text{Shift}} + \boxed{\text{Z}}$	Otáčení na rovině XY. Jinými slovy také pohled podél osy Z aktuálně vybraného zdroje (úhlu).
$\boxed{\text{N}}$ nebo $\boxed{[}$	Skok na předchozí timepoint.
$\boxed{\text{M}}$ nebo $\boxed{]}$	Skok na následující timepoint.
$\boxed{\uparrow}$, $\boxed{\downarrow}$	Přiblížení, resp. oddálení.

4.4 Renderování s použitím ImgLib2

Jak již bylo zmíněno BigDataViewer je prohlížeč, který umožňuje prohlížet sekvence nasnímaných obrázků. Obrázky jsou umístěny do společného globálního prostoru a je zobrazován libovolný řez tímto globálním prostorem. Pro každý renderovaný pixel na obrazovce je potřeba určit voxel ze zdrojových dat. Pro zavedení pojmu voxel budeme citovat ze zdroje [22]: „*Voxel označuje v počítačové grafice částici objemu představující hodnotu v pravidelné mřížce 3D prostoru. Jde o analogii k pixelu, který reprezentuje hodnotu v mřížce 2D prostoru. Voxely se používají nejčastěji při vizualizaci a analýze lékařských a vědeckých dat*“. Intenzita daného voxelu musí být konvertována z prostoru, kde byla definovaná (nasnímaná) do RGB barev pro zobrazení na obrazovce počítače. Tyto operace jsou na pozadí prováděny knihovnou ImgLib2.

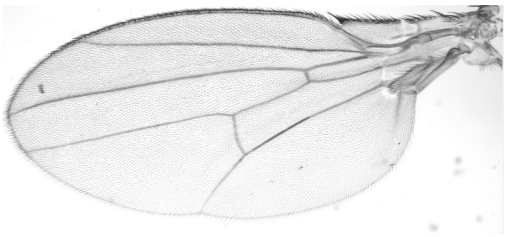
ImgLib2 je open-source framework pro manipulaci s n-dimenzionálními daty (obrázky) napsaný v programovacím jazyce Java. Skrze obecná rozhraní vytváří abstrakci nad dimenzionalitou, typem obrázků a jejich ukládáním. Knihovna umožňuje psát znovu použitelný kód, který je nezávislý na zmíněných vlastnostech. Mezi další cíle patří oddělení vývoje algoritmu a nutnost správy dat nebo rozšířitelnost knihovny (nové typy pixelů, optimálnější strategie ukládání apod.) Obrázek je tímto frameworkem definován jako libovolné mapování z podmnožiny n-dimenzionálního euklidovského prostoru do hodnoty obecného typu „pixel“ [4]. Mezi klíčové funkce ImgLib2, které využívá BigDataViewer patří:

- Virtualizovaný přístup k pixelu.
- Virtualizované rozšíření obrázku.
- Virtualizovaná interpolace obrázku.
- Virtualizovaná transformace souřadnic.

Nyní si popíšeme, co přesně každá odrážka znamená. Začněme s virtualizovaným přístupem k pixelu což představuje úplný základ.

4.4.1 Virtualizovaný přístup k pixelu

Virtualizovaný přístup znamená, že se k pixelům obrázku přistupuje skrze abstraktní vrstvu, která skrývá informace o tom, jak je obrázek uložen. To umožňuje využívat libovolného mechanismu pro ukládání obrázků. Jedním takovým je *CellImg* na kterém je BigDataViewer postaven. *CellImg* implementuje přímo ImgLib2 knihovna. Jedná se o kontejner, který ukládá n-dimenzionální obrázek jako množinu polí a každé pole představuje individuální n-dimenzionální blok obrázku. Virtualizovaný přístup může například skrývat logiku, kdy každý pixel je uložen jako prvek v poli. Individuální bloky obrázku jsou do paměti nahrávány podle potřeby. Spouštěčem nahrávání je právě virtualizovaný přístup k pixelu. Pokud algoritmus přistupuje k pixelu, který není v paměti, pak je



Obrázek 4.2: Křídlo octomilky

Obrázek 4.3: Křídlo octomilky - obrázek byl rozšířen v ose x za použití techniky zrcadlení. Nalevo od vyznačené přerušované čáry je oblast, kde se nachází původní obrazová data. Napravo od ní začíná doplnění zrcadlením.

odpovídající blok obrázku nahrán, a navíc uložen do mezipaměti (cache). Takový způsob je velmi výhodný pro renderovací algoritmy, které tak můžou pracovat s myšlenkou, že všechny data jsou uloženy v operační paměti po celou dobu.

4.4.2 Virtualizované rozšíření obrázku

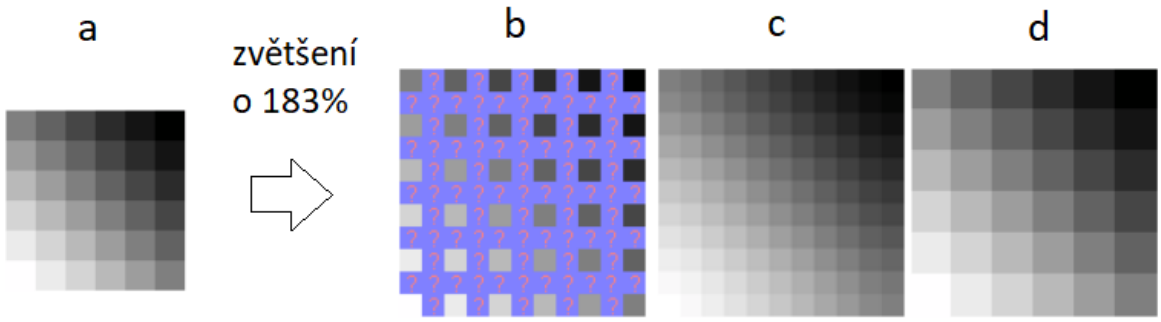
Virtualizované rozšíření obrázku znamená, že obrázek je rozšířen do nekonečna a my definujeme, jak se mají počítat hodnoty, které jsou za hranicemi obrázku. Můžeme například definovat pravidlo, podle kterého bude původní obrázek rozšířen a bude se zrcadlit do nekonečna, viz obrázky 4.2 a 4.3. Virtualizovaný přístup zajistí generování pixelů mimo hranice obrázku. Poznamenejme, že renderovací algoritmus nemusí brát v úvahu, zdali přistupuje ještě uvnitř nebo mimo hranice obrázku.

4.4.3 Virtualizovaná interpolace obrázku

ImgLib2 poskytuje tři interpolační schémata: *Nearest-neighbor* (Nejbližší soused), *Linear* (Lineární) a *Lanczos* (Lanczosova interpolace). BigDataViewer používá jako výchozí metodu nejbližší soused [13], kterou si společně s pojmem interpolace popíšeme.

Interpolace

Pro definici pojmu interpolace odkážeme na zdroj [23]. Představme si funkci $f(x)$, která je zadaná pomocí tabulky. Tzn. že máme tabulku, kde je pro nějaký výčet hodnot vždy uvedena hodnota



Obrázek 4.4: Interpolace bitmapového obrázku. Upraveno dle [24]. **Obrázek a)** představuje původní obrázek. **Obrázek b)** představuje zvětšený původní obrázek, ale bez použití metody interpolace (chybějící pixely značí otazníky). **Obrázek c)** znázorňuje zvětšený obrázek s použitím interpolace (je dosaženo vyššího rozlišení). Pro srovnání zobrazuje **obrázek d)** zvětšený původní obrázek pouze pomocí přiblížení (zooming).

funkce. Úlohou interpolování se myslí sestrojení takové funkce $\phi(x)$, která souhlasí s funkcí $f(x)$ v bodech uvedených v tabulce a v ostatních bodech nějakého intervalu (a, b) z definičního oboru funkce $f(x)$ je (s určitou přesností) přibližně rovna funkci $f(x)$.

Jinými slovy, interpolace je založena na výpočtu mezilehlých hodnot bodů z hodnot původního obrázku, které leží v okolí. Často se realizuje při zvětšení, resp. zmenšení rozlišení obrázku. Takto vznikne nové, digitálně větší rozlišení původního obrázku. Obrázek 4.4 znázorňuje interpolaci bitmapového obrázku. Poznamenejme, že i když interpolace umožňuje digitálně zvětšit obrázek, snaží se pouze napodobit skutečnost za cenu ztráty kvality a zubatosti (*aliasing*).

Na konec této sekce si krátce popíšeme metodu zvanou nejbližší soused (nearest-neighbor), jenž BigDataViewer používá jako výchozí metodu interpolace. Nicméně, není to jediná metoda, kterou BigDataViewer nabízí. Kromě metody nejbližší soused nabízí také *tri-linear* metodu interpolace. Vraťme se ale k metodě nejbližší soused. Jde o metodu, která je považována za nejjednodušší způsob interpolace. Spočívá v okopírování hodnoty nacházející se nejblíže požadované hodnotě. Mezi nedostatky této metody patří vynechávání tenkých čar při zmenšení obrázku. Naopak při zvětšení dojde k projevu rozmazání původního obrázku a ten pak vypadá, jako by byl složen z velkého množství čtverečků. Taktéž dochází ke značnému aliasingu (zubatosti).

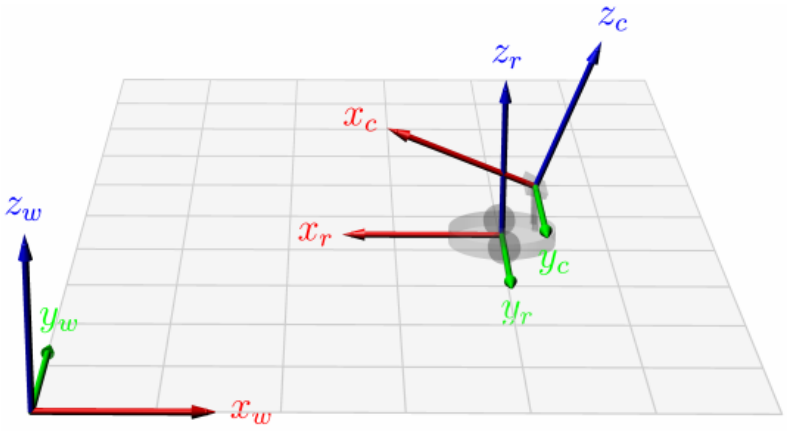
4.4.4 Virtualizovaná transformace souřadnic

Virtualizovaná transformace souřadnic se používá pro transformace obrázků do jiného souřadnicového systému. O souřadnicových systémech si řekneme více v další podkapitole. Transformace souřadnic je prováděna „on-the-fly“ (volně přeloženo: „*za běhu*“), až když se přistupuje k pixelu. To znamená, že se nekopírují data původního obrázku, ale přístup na pixel transformovaného obrázku znamená přístup na odpovídající pixel původního obrázku.

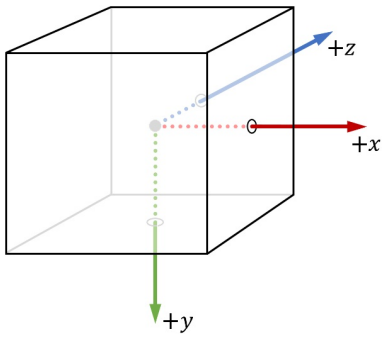
4.4.5 Souřadnicový systém

Souřadnicový systém úzce souvisí s pozicí a orientací objektu v prostoru. Jakýkoliv popis pozice objektu v prostoru musí být vždy popsán ve vztahu k nějakému souřadnicovému systému. Není například příliš užitečné vědět, že se nějaký objekt nachází na pozici $(-10, 20, 15)$, pokud nevíme kde se nachází počátek souřadnicového systému, tj. pozice $(0, 0, 0)$, a v jakém směru míří osy x, y, z. Zkusme si popsat podstatu souřadnicových systému na reálném příkladu.

Představme si robota, který se pohybuje po místnosti a má na sobě pohyblivou kameru. Pokud zachytí kamera nějaký libovolný objekt, může nám dát informaci: „nalezen objekt, nachází se 1 metr před kamerou“. Uživatel bude ale pravděpodobně chtít vědět kde se nachází daný objekt vzhledem k místnosti. Můžeme tedy definovat tři souřadnicové systémy, jak je tomu na obrázku 4.5. Souřadnicový systém označený indexem w značí místnost samotnou. Je pevně umístěný na nějakém známém místě v prostoru a nehýbe se s ním. Souřadnicový systém robota je označený indexem r. Můžeme si představit, že tento souřadnicový systém je připevněn na základně robota a pohybuje se tak s ním. Jinými slovy, robot se vždy nachází na pozici (0,0,0) v jeho vlastním souřadnicovém systému. Stejným způsobem můžeme interpretovat souřadnicový systém označený indexem c, který je připevněn ke kameře. Nakonec, pomocí translace mezi jednotlivými souřadnicovými systémy můžeme získat pozici objektu v souřadnicovém systému w, tedy pozici objektu v místnosti [25]. Poznamenejme ještě, že v této práci budeme využívat pravoúhlý souřadnicový systém neboli pravidlo pravé ruky, jak znázorňuje obrázek 4.6.



Obrázek 4.5: Zobrazení tří souřadnicových systémů v 3D prostoru. Zdroj: [25].



Obrázek 4.6: Souřadnicový systém - pravidlo pravé ruky. Zdroj: [26]

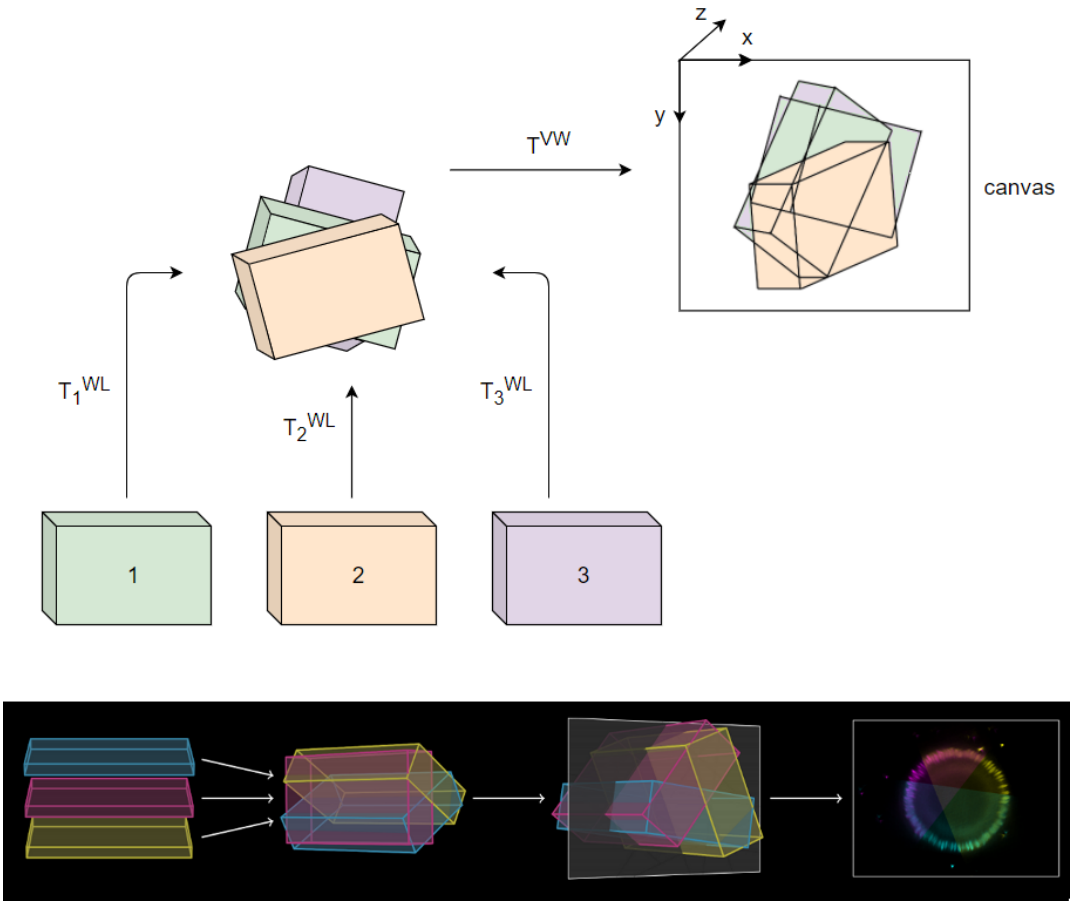
4.5 Renderovací algoritmus BigDataVieweru

Tato podkapitola popisuje základní renderovací algoritmus používaný v BigDataVieweru pro prohlížení jednotlivých řezů. Dle [13] zavedeme tři souřadnicové systémy (viz obrázek 4.7):

1. Lokální souřadnicový systém L bude definovaný souřadnicemi 3D voxelů z nasnímaných obrázku. Tyto obrázky jsou uloženy do svazků, tzv. *image volumes*.
2. Globální souřadnicový systém W definovaný jako libovolný 3D izotropní souřadnicový systém.
3. A nakonec souřadnicový systém prohlížeče V. Ten bude definovaný jako 3D souřadnicový systém tak, že jeho rovina z = 0 bude shodná s vykreslovacím plátnem na obrazovce (canvas). To znamená, že například hodnota pixelu na souřadnici (5,7,0) v souřadnicovém systému V bude renderována jako pixel na souřadnici (5,7) v canvas.

Dále označíme T^{WL} jako transformaci lokální souřadnice x^L do globální souřadnice x^G a T^{VW} jako transformaci z globálního souřadnicového systému W do souřadnicového systému prohlížeče V.

Algoritmus 1 popisuje pomocí pseudokódu základní algoritmus pro renderování v BigDataVieweru. Kód můžeme interpretovat takto: každý svazek obrázků je virtuálně rozšířen a interpolován. Virtuální interpolace (kapitola 4.4.3) je v pseudokódu znázorněna jako *interpolate(...)* a virtuální rozšíření (kapitola 4.4.2) jako *extend(...)*. Následně jsou souřadnice virtuálně transformovány do globálního souřadnicového systému s využitím transformace T^{WL}, resp. do souřadnicového systému prohlížeče V za pomocí T^{VW} transformace. Poté se (virtuálně) konvertují pixely z obecného typu *pixel* (definovaného knihovnou ImgLib2) do pixelu typu RGB tak, aby jej bylo možné zobrazit. Nakonec můžeme RGB pixely jednoduše přečíst z roviny z = 0 a s použitím tzv. *blending funkce* kombinovat a vykreslit na canvas. Jako *blending function* je v současné době v BigDataVieweru použita suma [13].



Obrázek 4.7: Transformace při vykreslování řezů ze svazků. Upraveno dle [13].

Předpokládáme následující vstupy pro renderovací algoritmus 1 (popsán pseudokódem):

- Renderovací transformace T^{VW} - mapování z globálního souřadnicového systému do souřadnicového systému prohlížeče.
- n zdrojových svazků (volumes) $I_1, \ldots, I_n$ s uloženými daty typu pixel $\tau_1, .., \tau_n$ - např. obrazová data z BigDataServeru ve formátu HDF5 a datovým typem pixelů *unsigned short*.
- n transformací $T_1^{WL}, \ldots, T_n^{WL}$, kde T_i^{WL} mapuje souřadnice voxelu v I_i do souřadnice globálního souřadnicového systému.
- n konvertorů $C_1, ..., C_n$, kde konvertor C_i je funkce pro mapování hodnot z typu pixel τ_n do RGB pixelů.
- 2D RGB canvas O pro renderování pixelů na obrazovce.

Algoritmus 1: Základní renderovací algoritmus. Zdroj: [13].

Vstup: Renderovací transformace T^{VW},
Zdrojové svazky $I_1, ..., I_n$,
Transformace dat $T_1^{WL}, ..., T_n^{WL}$,
Konvertory dat $C_1, ..., C_n$,
Canvas O

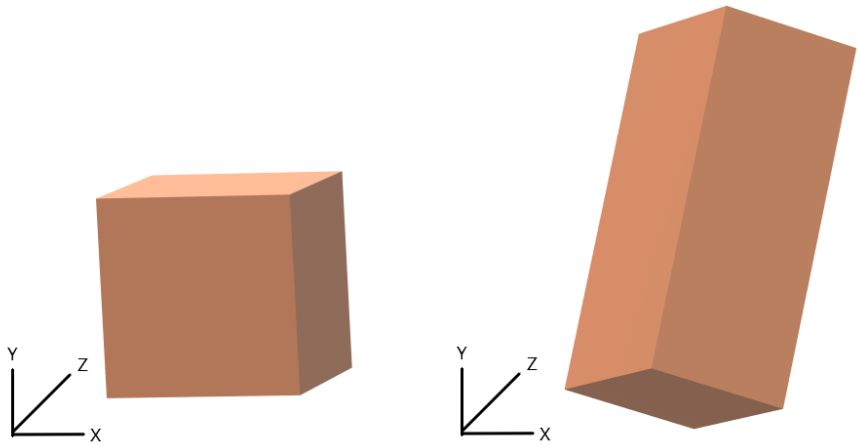
Výstup: Renderovaný obrázek v O

for $i \in \{1, ..., n\}$ **do**
  $I_i^{'} := interpolate(extend(I_i))$
  $I_j^{''} := T^{VW}(T_i^{WL}(I_i^{'}))$
  $J_i := C_i(I_j^{''})$
end
for ***every canvas pixel*** $(x, y) \in O$ **do**
  set $O(x, y) := B_{i=1}^{n} J_i(x, y, 0)$
end

Vizualizovaná data pochází z více zdrojů. Každý takový zdroj poskytuje jeden 3D obrázek pro určitý bod v čase *(timepoint)*. Pozorovaný vzorek je snímán z více úhlů, kde každý úhel snímaní je jeden zdroj dat.

4.6 Transformace obrázků

Transformace obrázků jsou operace, která nám umožňují s obrázkem dále pracovat. Např. rotovat s ním ve 2D nebo ve 3D, měnit měřítko, posouvat ho atd. Příklad takových operací si můžeme ukázat na obrázku 4.8. Jednou z možností, jak provádět transformace obrázků je použitím afinních transformací. Ty si teď popíšeme.



Obrázek 4.8: Na obrázku vlevo je vidět původní model krychle. Vpravo je vidět transformovaný model (s použitím škálování, rotace a translace.)

4.7 Afinní transformace

Úvodní definice převezmeme z [27]: „*Afinní transformace je zobrazení bodů jednoho afinního prostoru do jiného afinního prostoru*“. Začněme popsáním, co je to afinní prostor. Jedná se o prostor, ve kterém se vyskytují body. Tyto body je navíc možné jednoznačně identifikovat pomocí souřadnic, např. vektorů. Z toho vyplývá, že k tomuto prostoru musí být dán nějaký přidružený vektorový prostor a zobrazení, které ke každé dvojici z prostorů bodů přiřazuje prvek z přidruženého vektorového prostoru. Daný vektorový prostor určuje také dimenzi afinního prostoru. Předpokládejme trojrozměrný afinní prostor a v něm bod $\mathbf{X}$. Souřadnice bodu $\mathbf{X}$ nebo přesněji afinní souřadnice jsou definovány vektorem o třech prvcích $x = (x_1, x_2, x_3)$.

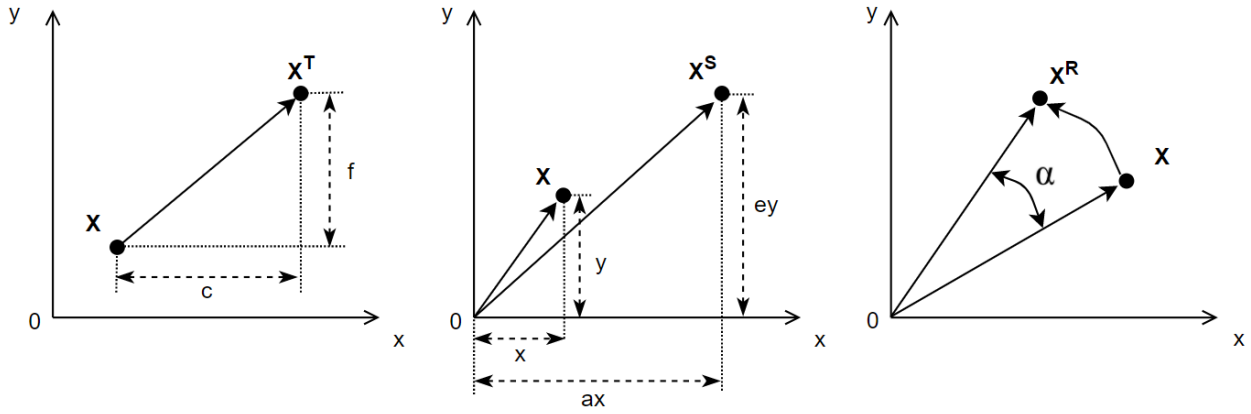
Pojďme si nyní ukázat několik afinních transformací ve 2D prostoru, který je mnohem jednodušší pro ilustraci. Později popíšeme nejběžnější afinní transformace ve 3D prostoru. Určeme si bod $x = (x, y)$. Afinní transformace bodu x jsou všechny transformace, které můžou být napsány následovně:

$$x' = \begin{bmatrix} ax + by + c \\ dx + ey + f \end{bmatrix},$$

kde a, b, c, d, e, f jsou skaláry [28].
Zkusme nyní pomocí různých hodnot skalárů pozorovat, jak se bude měnit matice x' a jaké to bude mít důsledky (znázorňuje také obrázek 4.9).

- Pokud bude $a = 1$, $e = 1$ a $b = 0$, $d = 0$, získáme translaci $x^T = \begin{bmatrix} x + c \\ y + f \end{bmatrix}$.

- Pokud bude $b = 0$, $d = 0$, $c = 0$, $d = 0$, získáme škálování (scaling) $x^S = \begin{bmatrix} ax \\ ey \end{bmatrix}$.
- A pokud bude $a = \cos\alpha$, $e = \cos\alpha$, $b = -\sin\alpha$, $d = \sin\alpha$, $c = 0$ a $f = 0$ dostáváme rotaci proti směru hodinových ručiček $x^R = \begin{bmatrix} x\cos\alpha - y\sin\alpha \\ x\sin\alpha + y\cos\alpha \end{bmatrix}$.



Obrázek 4.9: Ukázka translace (vlevo), škálování (uprostřed), rotace (vpravo)

Celkem jsme si ukázali 3 základní transformace: translaci - posouvá množinu bodů o fixní vzdálenost na osách x a y, škálování - škáluje množinu bodů nahoru nebo dolů ve směru osy x a y, rotace - otáčí množinou bodů vůči původní pozici. Poznamenejme, že pouze škálování ze zmíněných operací mění tvar, který je dán množinou bodů.

Afinní transformace jsou vlastně lineární transformace a jako takové mohou být reprezentovány jako násobení matice a vektoru. Neboli vztahem 4.1.

$$x' = Mx \tag{4.1}$$

Kde M je matice transformace, x je bod reprezentován vektorem a x' je transformovaný bod. Vztah můžeme také rozepsat jako:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} ax + by \\ dx + ey \end{bmatrix} = \begin{bmatrix} a & b \\ d & e \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

Výhoda takové maticové reprezentace je, že můžeme sérii komplexních transformací přepsat do jednodušších transformací. Mějme například nějaký objekt reprezentovaný množinou bodů a budeme chtít provést translaci, škálování a rotaci. Definujme si matici T pro translaci, matici S pro škálování a matici R pro rotaci. Dále si vytvořme sekvenci tří transformačních operací: 1. operace - translace, 2. operace - škálování a 3. operace - rotace. Tyto operace můžeme zapsat jako rovnici 4.2.

$$x' = T(S(R(x))) \tag{4.2}$$

Při násobení matic můžeme využít asociativní zákon a vynásobit matice T, S a R mezi sebou. Vznikne nám nová matice $M = TSR$ a celou transformaci pak můžeme zapsat jako 4.3.

$$x' = (TSR)x = Mx \tag{4.3}$$

Pokud budete mít komplexní model a budeme muset tedy transformovat tisíce bodů, bude výpočetně mnohem méně náročné udělat takovou operaci pouze jednou než s každou transformační maticí T, S, R zvlášť. Na závěr můžeme sepsat tyto získané matice lineární transformace:

Škálování: $\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$, Rotace: $\begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}$,

kde s_x a s_y škálují x a y souřadnice bodu, α je úhel rotace proti směru hodinových ručiček. Pozorný čtenář si všimne, že jsme zapomněli napsat matici pro translaci. Kdybychom se o to pokusili, museli bychom vytvořit např. následující výpočet:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Jenže takové násobení matic není platná operace vzhledem k maticovým operacím. Celý problém je v tom, že translace není lineární transformace. Tento problém si více rozebereme v další podkapitole.

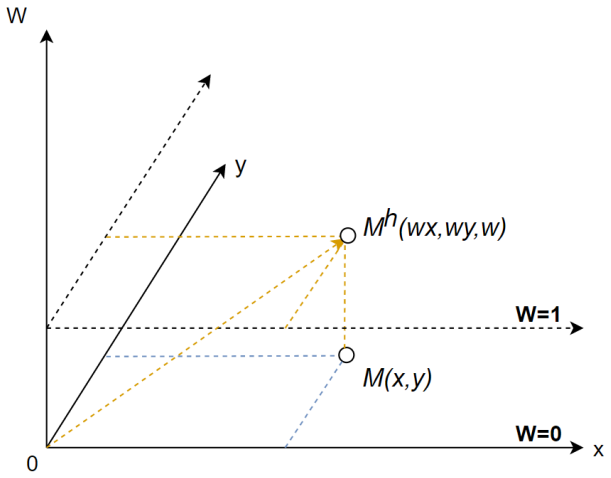
4.7.1 Homogenní souřadnice

Před dalším popisováním transformací je nutné zavést a osvětlit pojem homogenní souřadnice. Je to způsob, jakým se zadávají souřadnice a umožňuje provádět transformační výpočty pomocí maticových operací. Představme si bod $M(x, y)$, kde x a y reprezentují souřadnice bodu. Nyní si ho zkusíme reprezentovat v homogenních souřadnicích jako bod $M^h(wx, wy, w)$, kde w je různé od nuly. Následně bod s homogenními souřadnicemi (X, Y, W) má kartézské souřadnice $x = X/W$ a $y = Y/W$ [29]. Reprezentaci bodu M v homogenních souřadnicích ukazuje obrázek 4.10.

Nyní si ukážeme, jak můžeme pomocí homogenních souřadnic vyřešit problém, kdy operace translace není lineární transformace. Řešením je tento 2D problém převést na 3D problém. Vezmeme všechny naše body $X = (x, y)$, které vyjádříme jako 2D vektor a převedeme je do 3D vektoru. Přitom nastavíme třetí souřadnici u všech bodů na 1. Tedy:

$$\begin{bmatrix} x \\ y \end{bmatrix} => \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Abychom odlišili tuto třetí souřadnici od typické z souřadnice, nazveme ji dle definice výše jako w. Podobným způsobem rozšíříme naše matice lineární transformace z předchozí kapitoly tak, že přidáme jeden extra řádek a sloupec:



Obrázek 4.10: Reprezentace bodu M v homogenních souřadnicích

Škálování: $\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$, Rotace: $\begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$.

Pojďme si ukázat, co se stane, pokud nyní vynásobíme 3D homogenní matici a 3D vektor:

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax+by \\ cx+dy \\ 1 \end{bmatrix}$$

Jak si můžeme všimnout, výsledek je stejný jako ve 2D, až na třetí w souřadnici, která zůstává rovna 1. Všechno, co jsme museli udělat, je umístit náš 2D bodu do 3D roviny ($w = 1$). Nad touto 3D rovinou můžeme provádět další transformace, ale operace zůstávají stále 2D „kompatibilní“.

Zpět k našemu problému s translací. Upravme 3D homogenní matici tak, že umístíme translační parametry do třetího sloupce. Dostaneme:

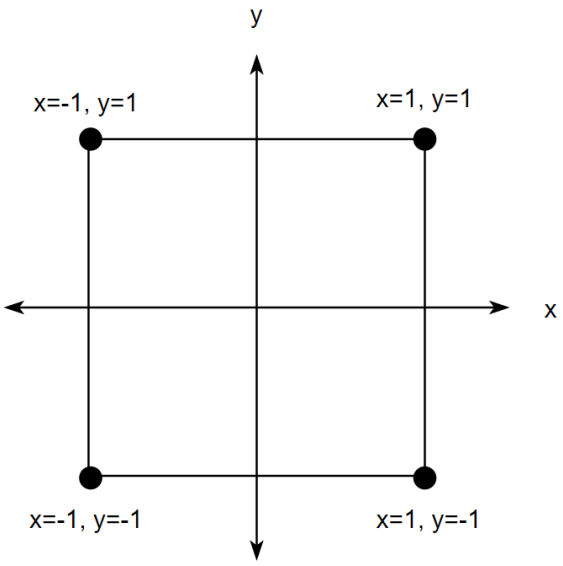
$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax+by+c \\ dx+ey+f \\ 0+0+1 \end{bmatrix}$$

Nyní můžeme konečně provádět translaci jako lineární transformaci s použitím homogenních souřadnic. Nakonec můžeme napsat finální podobu matice pro translaci.

Translace: $\begin{bmatrix} 1 & 0 & \delta x \\ 0 & 1 & \delta y \\ 0 & 0 & 1 \end{bmatrix}$, kde δx je posun ve směru osy x, resp. δy je posun ve směru osy y.

4.7.2 Příklad transformace s použitím transformačních matic

V této sekci si transformační matice a homogenní souřadnice ukážeme v praxi na příkladu. Řekněme, že budeme mít čtverec velikosti 2x2, který je umístěný tak, že jeho střed je uprostřed souřadnicového systému (jak je zobrazeno na obrázku 4.11). Tento čtverec budeme chtít otočit o 45° okolo jeho středu a následně posunou tak, aby byl jeho střed na souřadnicích x = 3 a y = 2.



Obrázek 4.11: Čtverec 2 x 2

S využitím transformačních matic dostaneme tuto výslednou transformační matici:

$$M = T_{(3,2)}R_{45} = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 45 & -\sin 45 & 0 \\ \sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos 45 & -\sin 45 & 3 \\ \sin 45 & \cos 45 & 2 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 & 3 \\ \sqrt{2}/2 & \sqrt{2}/2 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

Nyní provedeme transformaci bodů čtverce.

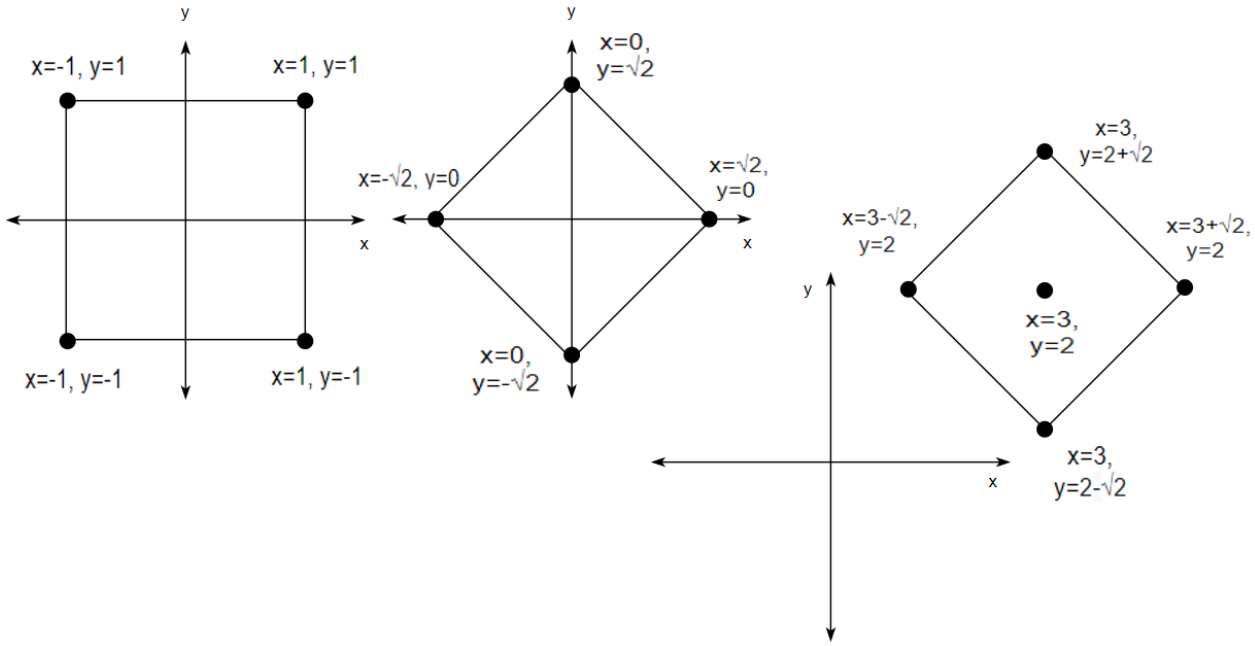
$$M \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 & 3 \\ \sqrt{2}/2 & \sqrt{2}/2 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 2+\sqrt{2} \\ 1 \end{bmatrix},$$

$$M \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 & 3 \\ \sqrt{2}/2 & \sqrt{2}/2 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3-\sqrt{2} \\ 2 \\ 1 \end{bmatrix},$$

$$M \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 & 3 \\ \sqrt{2}/2 & \sqrt{2}/2 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3+\sqrt{2} \\ 2 \\ 1 \end{bmatrix},$$

$$M \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 & 3 \\ \sqrt{2}/2 & \sqrt{2}/2 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 2-\sqrt{2} \\ 1 \end{bmatrix}.$$

Grafické znázornění transformací ukazuje následující obrázek 4.12.



Obrázek 4.12: Transformace čtverce - znázornění transformací

4.8 Afinní transformace ve 3D prostoru

Všechny naše poznatky z předchozích kapitol můžeme aplikovat i na 3D prostor. To znamená převést všechny naše 3D body do homogenních souřadnic a nastavit čtvrtou (w) souřadnici na 1:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} => \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Dále pak můžeme použít tyto matice pro 3D afinní transformace (opět v homogenní formě):

Translace: $\begin{bmatrix} 1 & 0 & 0 & \delta x \\ 0 & 1 & 0 & \delta y \\ 0 & 0 & 1 & \delta z \\ 0 & 0 & 0 & 1 \end{bmatrix}$, Škálování: $\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$.

Dle [28] je rotace ve 3D prostoru možná třemi různými způsoby:

$$\text{Rotace kolem osy x: } \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$\text{Rotace kolem osy y: } \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$\text{Rotace kolem osy z: } \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Úhly α, β, γ jsou úhly rotace kolem os a jsou to tzv. Eulerovy úhly. Poznamenejme, že pořadí, v jakém jsou rotace prováděny je velmi důležité a musí být na začátku specifikováno. Vyplývá to z pravidel maticových operací. Afinní transformace jsou asociativní, ale ne komutativní. Znamená to tedy, že výsledek transformace tří rotací $R_\alpha R_\beta R_\gamma$ bude jiný oproti výsledku rotací $R_\gamma R_\beta R_\alpha$.

Afinní transformace popsané výše jsou základním kamenem operací, které provádí BigDataViewer při zobrazovaní dat. Parametry právě těchto transformačních rovnic můžeme nalézt v XML konfiguračním souboru datasetu.

Kapitola 5

Aplikace pro zobrazování dat v rámci webového prohlížeče

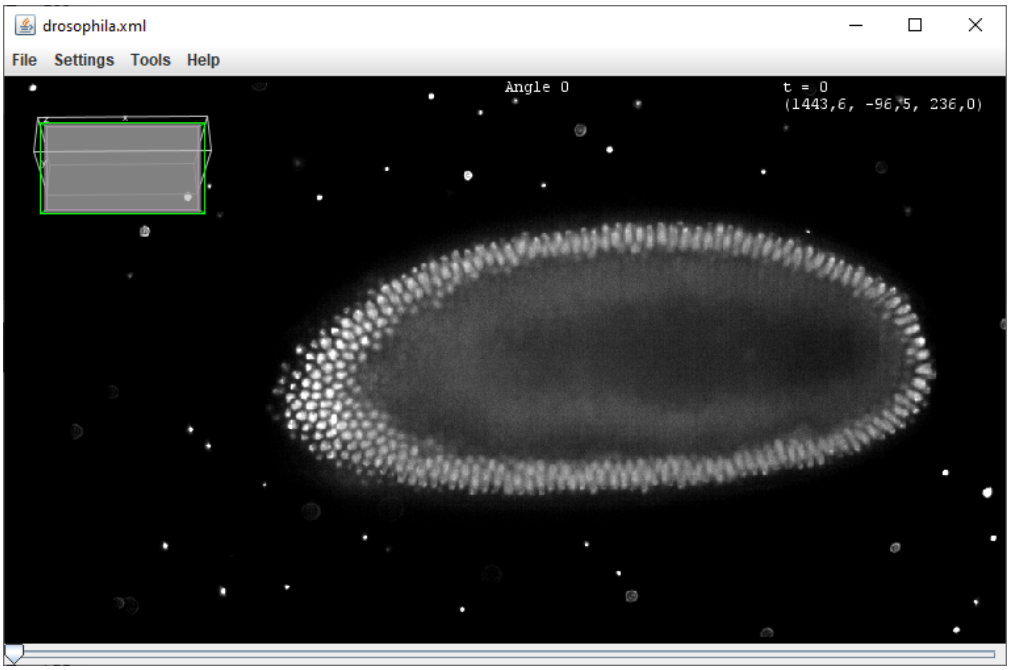
V předchozích kapitolách práce jsme se věnovali teoretické stránce. Následující kapitoly budou věnovány implementaci aplikace.

5.1 Úvod

Cílem je vytvořit webovou aplikaci, pomocí které by se daly v reálném čase prohlížet obrazová data z BigDataServeru. Aplikace by měla být schopna napojení na libovolný BigDataServer pomocí standardního protokolu HTTP. Do jisté míry by měla obsahovat podobný vizuální styl a stejnou funkcionalitu jako Fiji BigDataViewer (viz obrázek 5.1) tak, aby uživatelé BigDataVieweru byli schopni bez vetší potřeby studování manuálu používat i tuto aplikaci.

Jeden z hlavních případu užití je použití aplikace jako tzv. plugin. Tedy vsazení do webové stránky jako interaktivní prvek umístěný vedle textu (např. <iframe> element). Uživateli se tak propojí studovaný text s reálnou ukázkou. Aplikace by měla fungovat i jako samostatná webová stránka tak, aby poskytovala lepší práci s pozorovaným vzorkem. V první verzi by měla aplikace poskytovat alespoň základní operace s obrazovými daty jako translace, rotace apod.

Návrh aplikace by měl počítat s možností škálování a dalšího rozšíření funkcionality do budoucna. Všechny požadavky a cíle ještě shrneme v následující sekci.



Obrázek 5.1: Fiji BigDataViewer

5.2 Cíle

Základní cíle implementace můžeme shrnout v těchto šesti bodech:

1. Vytvoření webové aplikace pro prohlížení obrazových dat z BigDataServeru.
2. Aplikace musí být schopna napojit se na libovolný BigDataServer a získat z něj data.
3. Aplikace musí poskytovat základní operace nad obrazovými daty (translace, rotace apod.)
4. Bude možné upravovat jas a kontrast obrazových dat.
5. Aplikace musí být kompatibilní s co největší škálou webových prohlížečů.
6. Aplikace bude škálovatelná.

5.3 Architektura

Samotná aplikace se skládá ze dvou samostatných komponent: **bd-image-viewer** (front-end část) a **bd-image-processor** (back-end část). Důvodem pro rozdělení do dvou komponent bylo logické oddělení části pro zobrazování dat (prezentační vrstva) a pro funkční část (aplikační vrstva) tak, jak je zvykem při vývoji *enterprise* aplikací. Také bylo potřeba vzít v potaz to, jakým způsobem jsou obrazová data uložena a poskytována skrze webové rozhraní BigDataServeru.

Z kapitoly 3 víme, že obrazová data jsou poskytována jako oddělené celky dat, které je potřeba seskupit a provést nad nimi inicializační operace (základní transformace, korekce jasu apod.) Pro stahování dat z BigDataServeru se navíc předpokládá vícevláknový přístup. Vyplývá to přímo ze způsobu, jakým jsou data uložena ve formátu H5. Sekvenční stahováni by totiž mohlo pro větší obrazová data učinit aplikaci velmi pomalou až nepoužitelnou. To všechno byly důvody pro vytvoření komponenty bd-image-processor, která slouží jako prostředník mezi front-end komponentou a BigDataServerem. Důležitým aspektem bylo také využití funkcí knihovny ImgLib2, nad kterou je vlastně BigDataViewer postaven. Knihovně ImgLib2 budeme věnovat jednu z následujících podkapitol. Protože je knihovna ImgLib2 napsaná v programovacím jazyce Java, nabízel se logicky tento programovací jazyk i pro tvorbu komponenty bd-image-processor. Zároveň má Java dobrou podporu pro vícevláknové aplikace, je multiplatformní a umožňuje vytvoření HTTP serveru pro klientské aplikace. V neposlední řade má s ní autor dlouholeté zkušenosti.

Aplikace, respektive její front-end část je zamýšlena pro použití ve webových prohlížečích. To znamená použití technologií *HTML*, *CSS* a *JavaScript*. Na trhu dnes existuje mnoho knihoven pro tvorbu uživatelských rozhraní postavených na vyjmenovaných technologiích. Dle statistiky z webové stránky `www.statista.com` můžeme uvést první tři nejpoužívanější webové frameworky v roce 2020: **jQuery, React, Angular**[1]. Při výběru vhodného frameworku jsem zvažoval hlavně samotnou rychlost frameworku, jednodušší syntaxi, která by umožňovala používat přímo HTML tagy, snadnou udržovatelnost, testovatelnost a podporu pro použití dalších knihoven. Podle těchto požadavků jsem zvolil JavaScript framework React od společnosti Facebook. React si více popíšeme v následující podkapitole.

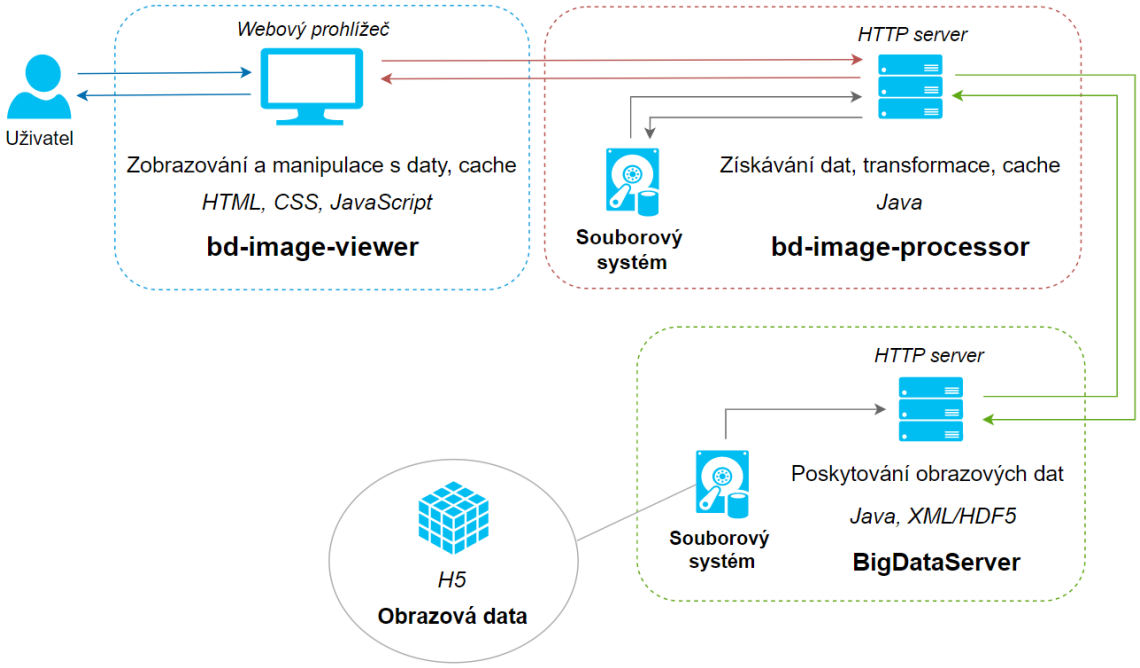
Zvolili jsme si technologie pro tvorbu aplikace a rozdělili jsme si aplikaci do dvou komponent. Nyní je potřeba navrhnout jakým způsobem budou spolu tyto dvě komponenty, tedy back-end a front-end, komunikovat. Zvolit vhodný způsob komunikace je klíčové pro budoucí rozšiřitelnost aplikace. Komunikace bude spočívat v posílání (transformovaných) obrázků z back-end komponenty na front-end. Výchozí formát obrázků bude PNG. Front-end bude zase posílat informace o tom, jak mají být obrázky transformovány. Jako komunikační protokol bude použitý standardní HTTP protokol. API bude definováno jako REST rozhraní, tzn. bude datově orientováno.

Poslední věcí, kterou je potřeba zmínit je ukládání dat do cache neboli mezipaměti. V této práci se budeme držet zavedených pojmů „cache“, „cachování“. Jelikož aplikace pracuje s obrazovými daty, která můžou být poměrně velká (řádově stovky megabajtů) a navíc transformační operace můžou být časově náročnější, je potřeba ukládat mezivýsledky do cache. Musíme tedy uvažovat o zavedení nějakého systému cachování do aplikace. Vzhledem k povaze dat se bude muset část z nich odkládat na souborový systém na disk, jinak by mohlo rychle dojít k vyčerpaní operační paměti určené pro cache. A část z nich (které jsou zrovna potřeba) bude dostupná v operační

[1]Kompletní statistika je dostupná na stránce `https://www.statista.com/statistics/1124699`

paměti. Strategii a detaily cachování si více popíšeme v kapitole 7.6.

Máme tedy definovanou architekturu aplikace. Jedná se o hrubý náhled na systém bez konkrétních detailů. Všechny důležité poznatky jsou shrnuty v obrázku 5.2. V angličtině se pro takový hrubý náčrt používá pojem „high-level overview“.



Obrázek 5.2: Architektura aplikace. Šipky znázorňují toky dat. Čárkovanou čarou jsou ohraničeny komponenty. Také je zde popsán hlavní význam komponenty a technologie (*text kurzívou*).

Kapitola 6

Implementace front-end části

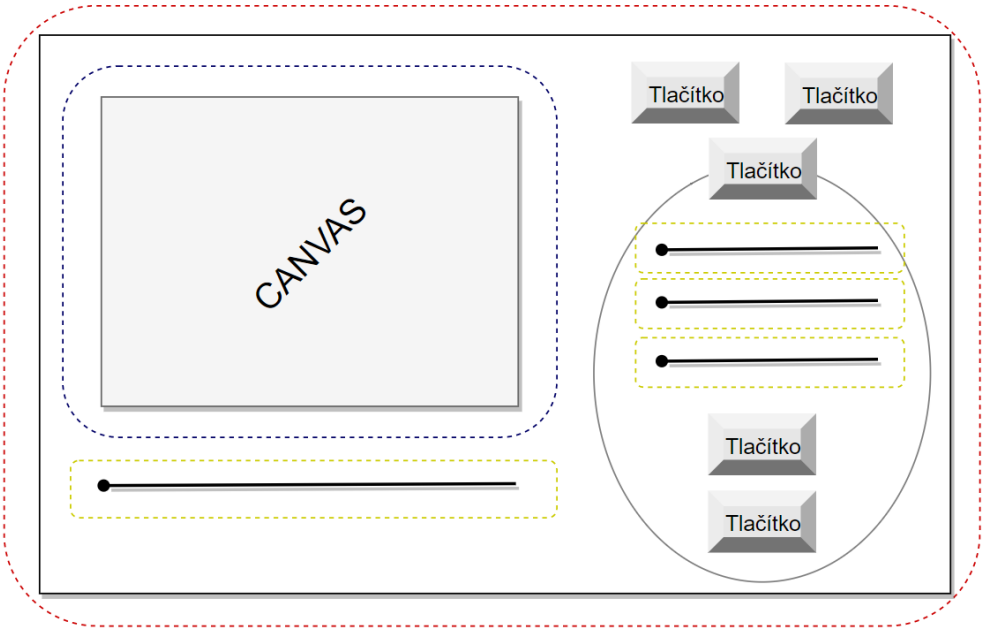
Front-end část bude implementována v React technologii. Pro vývoj budeme používat Node.js ve verzi 12.19.0 spolu se správcem JavaScript balíčku npm ve verzi 6.4.18.

6.1 React

React je JavaScript knihovna pro tvorbu webových uživatelských rozhraní. Oficiálně ho v roce 2013 představil Jordan Walke na konferenci JSConf US [30]. Na vývoji se podílí společnost Facebook spolu s komunitou samostatných vývojářů pod licencí MIT. React je postaven na komponentách, což jsou různé znovupoužitelné HTML elementy se zapouzdřenou funkcionalitou. Při psaní zdrojového kódu se nepoužívají HTML tagy jako takové, ale JSX (**J**ava**S**cript **X**ML). Ty se pak transformují do HTML kódu. Nicméně, syntax JSX tagů je velmi podobná HTML tagům. Pochopení principu komponent a jejich správné navržení je klíčové pro vývoj aplikací v Reactu. Každá komponenta má svůj vnitřní stav - „state“ a vlastnosti - „props“. Data se v Reactu předávají mezi komponentami jen jedním směrem - z jedné (nadřazené) komponenty na druhou (vnořenou). Komponenta je pak na základě změny vnitřního stavu automaticky překreslena. Překresluje se pouze ta komponenta, u které došlo ke změně vnitřního stavu, nikoliv všechny komponenty.

6.2 Design

Budeme vycházet z designu Fiji BigDataVieweru (obrázek 5.1). UI budou tvořit tři komponenty a budou na stránce rozmístěny dle návrhu na obrázku 6.1. V případě použití aplikace jako *iframe*, vložený do jiné stránky, budou ovládací pole umístěny pod canvas komponentou.



Obrázek 6.1: Návrh uživatelského rozhraní. Na obrázku jsou barevně označeny tři komponenty - hlavní komponenta aplikace (*červeně*), canvas (*modře*), ovládací pole (*žlutě*). Po kliknutí na určité tlačítko se zobrazí další ovládací pole.

6.3 Komponenty

Hlavní komponenta zapouzdřuje veškerou logiku a vnitřní stav. Kromě toho je také nadřazená všem ostatním komponentám a předává jim data. Komponenta bude zodpovědná za:

- Zpracování událostí jako pohyb myší, stisknutí tlačítka apod. a úpravu vnitřního stavu dle těchto událostí.
- Sestavování URL pro volání bd-image-processor API a předávání sestavené URL canvas komponentě.
- Získání metadat o datasetu - metadaty se rozumí počet dostupných timepoints, setups a názvy views.
- Zpracování povinných i nepovinných parametrů z URL při prvním vstupu uživatele.

Získávání parametrů ze vstupní URL nám dává určitou flexibilitu. Díky této parametrizaci jsme schopni měnit konfiguraci zobrazení komponenty za běhu bez nutnosti restartu. Také jsme schopni měnit cílovou adresu BigDataServeru, výběr dataset a další parametry pomocí změny v URL adrese.

Další komponentou je ovládací pole. Skrze tato ovládací pole se může uživatel posouvat mezi jednotlivými timepoint, nastavovat jas a kontrast. Jde o poměrně jednoduchou komponentu, jejíž zdrojový kód můžeme napsat přímo do tohoto textu - výpis 6.1.

Komponenta má několik vlastností (*props*): název ovládacího pole (*label*), tzv. callback funkci pro obsluhu při změně hodnoty (*onChange*), aktuální hodnotu (*value*), minimální, resp. maximální povolenou hodnotu (*min*, *max*) a krok s jakým se hodnota mění (*step*). *Min, max* a *step* mají výchozí hodnoty, pokud nejsou předány při volaní.

```
class Field extends Component {
    render() {
        const {label, onChange, value, min, max, step} = this.props;
        return (
            <label>
                <span>{label}</span>
                <input type="range" min={min ? min : 0} max={max ? max : 10}
                    step={step ? step : 0.01} value={value}
                    onChange={e => onChange(parseFloat(e.target.value))} />
                <span>{value}</span>
            </label>
        );}}
```

Výpis 6.1: Zdrojový kód komponenty ovládací pole

Tato ovládací pole budou na stránce společně s dalším nastavením ve výchozím stavu skryty. Uživatel si je bude moci zobrazit kliknutím na tlačítko „*Settings*“.

Nyní se dostáváme k srdci celého front-endu, a to canvas komponentě. Právě ta je zodpovědná za stažení obrazových dat z back-endu a jejich vykreslení. Jedním z cílů při sestavování požadavků byla možnost upravovat jas a kontrast. Jistou korekci jasu už provádí back-end při inicializačních operacích. Nicméně, hodila by se také možnost jemného upravovaní jasu a kontrastu přímo na stránce. V Reactu je k dispozici balíček **gl-react-contrast-saturation-brightness** jako rozšíření knihovny gl-react, který nabízí kombinaci efektů jasu, kontrastu a saturace (sytost barev). Efekt saturace nebudeme potřebovat. Knihovna gl-react nabízí také možnost psaní WebGL programů v Reactu.

6.4 WebGL - gl-react

Web Graphics Library (*zkráceně WebGL*) je JavaScript API pro renderování 2D a 3D grafiky v rámci webových prohlížečů. Je plně integrován s ostatními webovými standardy, díky čemuž může využívat grafickou akceleraci, používat fyziku a zpracovávat obrazová data jako součást webové stránky (používá HTML5 element canvas a DOM rozhraní). WebGL elementy můžeme tedy používat současně se standardními HTML elementy a vytvářet tak obsah stránky. WebGL programy tvoří obslužný kód a shader kód. Obslužný kód je napsaný v JavaScriptu, shader kód využívá programovacího jazyka OpenGL ES Shading language (*zkráceně GLSL*). GLSL má podobnou syntax jako C nebo C++. Shader zdrojový kód je vykonáván na grafické kartě počítače [31, 32].

Zkusme si nyní blíže popsat *ContrastSaturationBrightness* komponentu, kterou využíváme pro zobrazení obrazových dat s možností úpravy jasu a kontrastu. Její implementace se nachází v souboru bd-image-viewer/src/ContrastSaturationBrightness.js v příloze. Volání pak můžeme najít v souboru bd-image-viewer/app/src/App.js. Výňatek z kódu můžeme vidět ve výpisu 6.2. Předáváme ji hodnotu saturace a jasu, *onDrawCallback* callback funkci a taky URL pro stažení obrazových dat (skrze bd-image-processor API). Komponenta na pozadí stáhne a vykreslí obrazová data. Případně upraví jas a kontrast dle nastavení uživatele. Callback funkce *onDrawCallback* je zavolána, když jsou obrazová data stažena a překreslena.

```
1   <Surface width={727} height={400}
2       [...]
3       <ContrastSaturationBrightness
4           saturation={saturation}
5           brightness={brightness}
6           onDrawCallback={this.onDrawCallback}>
7           {url}
8       </ContrastSaturationBrightness>
9   </Surface>
```

Výpis 6.2: ContrastSaturationBrightness komponenta

Na pozadí vykreslování je využívána předdefinovaná komponenta *Node*. Jedná se o primitivní komponentu, která renderuje daný shader program a ukládá jeho výstup do framebufferu. Také je potřeba obalit *ContrastSaturationBrightness* komponentu do *Surface*. *Surface* provádí finální vykreslení pro danou implementaci WebGL.

6.5 První spuštění

Pro první spuštění aplikace bd-image-viewer je potřeba provést následující kroky. V kořenovém adresáři bd-image-viewer aplikace v příloze je potřeba spustit příkaz „npm install“. Tím nainstalujeme všechny závislosti a knihovny. Následně je potřeba provést sestavení příkazem „npm run compile“ (opět v kořenové složce). Po úspěšném sestavení se můžeme přepnout do složky „app“ a pomocí příkazu „npm run start“ spustit aplikaci. Případně můžeme opět použít příkaz „npm run build“ a vznikne nám produkční sestavení (release). Vzniklé sestavení je možné umístit na server a pomocí webového serveru nebo proxy serveru zpřístupnit. Připomínáme verze Node.js 12.19.0 a npm 6.14.8, jež byly použity pro testovanou verzi aplikace.

6.5.1 Vstupní URL

Při prvním vstupu je potřeba specifikovat vstupní URL. Ta má následující předpis:

```
scheme://host[:port]/?baseUrl=v1&bigDataServerUrl=v2&datasetName=v3
[&timepoint=v4][&setup=v5][&level=v6][&slice=v7][&rotate=v8][&translate=v9]
```

Tabulka 6.1 popisuje význam URL atributů a další podrobnosti.

Tabulka 6.1: Popis parametrů vstupní URL pro bd-image-viewer komponentu

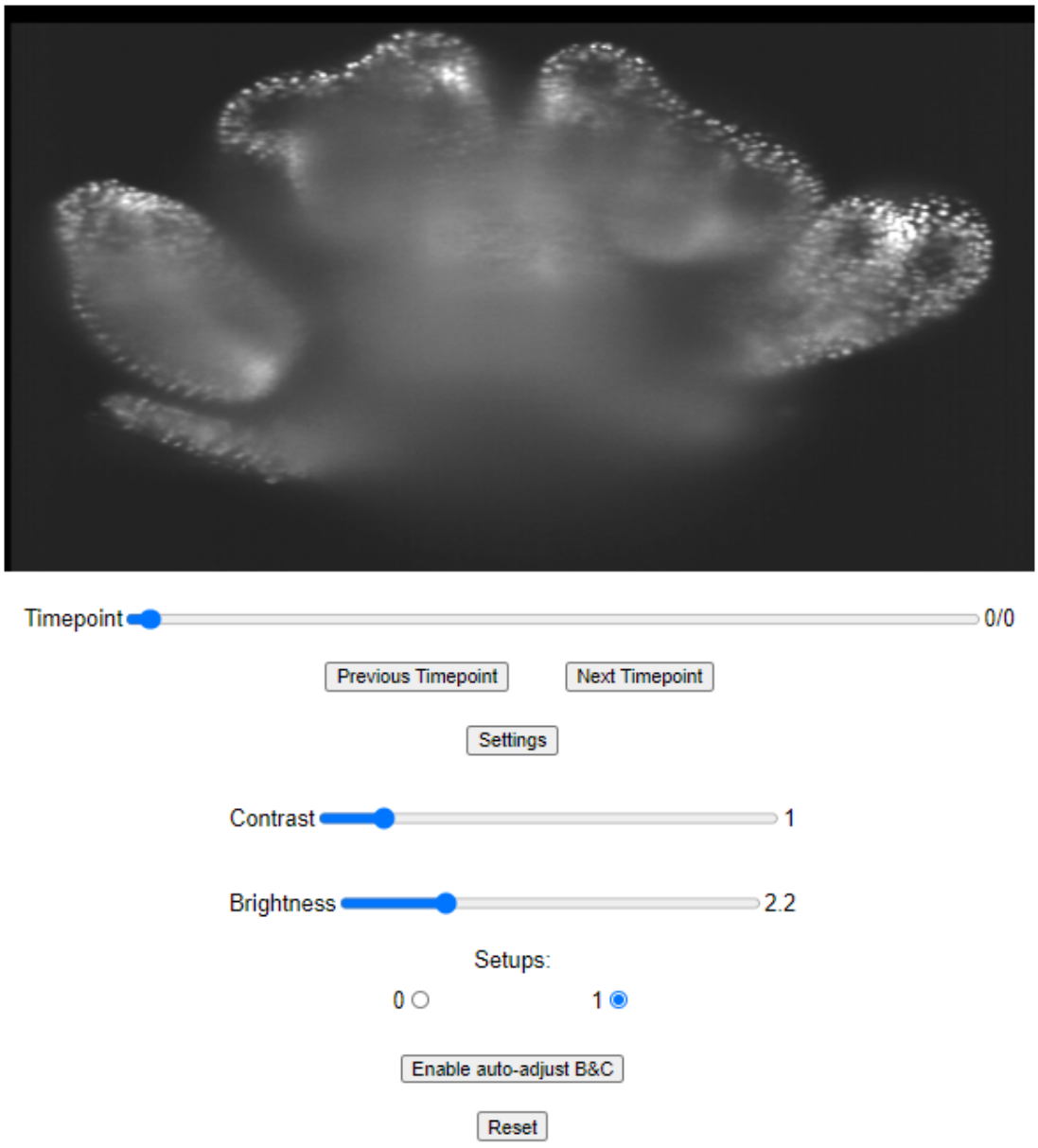
Parametr	Význam	Povinný	Výchozí hodnota
baseUrl	URL adresa pro *bd-image-processor* backend, např. `http://localhost:8080`.	Ano	
bigDataServerUrl	URL adresa BigDataServeru, např. `http://julius2.it4i.cz`.	Ano	
datasetName	Jméno požadovaného datasetu.	Ano	
timepoint	Číselná hodnota timepoint.	Ne	0
setup	Číselná hodnota setup id.	Ne	0
level	Číselná hodnota level.	Ne	2
slice	Číselná hodnota řezu.	Ne	1
rotate	Konfigurace rotace ve formátu „x:x_0_y:y_0“. Hodnoty x_0 a y_0 představují stupně. Např. „x:90“ značí rotaci okolo osy x o 90° po směru hodinových ručiček.	Ne	x:0_y:0
translate	Konfigurace translace ve formátu „x:x_0_y:y_0“. Hodnoty x_0 a y_0 představují vzdálenost v pixelech.	Ne	x:0_y:0

6.6 Navigace a ovládání webové aplikace

Ovládaní webové aplikace vychází z ovládání BigDataVieweru (kapitola 4.3). V současné verzi aplikace 0.1.0 je dostupné pouze ovládání myší nebo tlačítky na stránce. Ovládání myší je následující:

- Prohlížení řezů pomocí pohybu kolečka myši.
- Translace stisknutím pravého tlačítka myši a pohybem kurzoru po plátně.
- Rotace stisknutím levého tlačítka myši a tahem kurzoru po plátně ve směru rotace.

Další ovládání je dostupné skrze tlačítka. Tlačítky Previous Timepoint a Next Timepoint se posouváme na předchozí, resp. následující *timepoint*. Dostupné *timepoints* můžeme vidět pod plátnem. Také zde nalezneme tlačítko Settings, jenž zobrazujeme další ovládací pole jako nastavení kontrastu, jasu, volbu aktivního *setup*, případně povolení extra úpravu jasu a kontrastu tlačítkem Enable auto-adjust B&C. Posledním tlačítkem je Reset. Jeho funkcí je resetování všech nastavení do výchozího stavu (včetně rotace a translace). Ukázka aplikace včetně ovládácích prvků je na obrázku 6.2.



Obrázek 6.2: Ukázka navigace aplikace bd-image-viewer u

Kapitola 7

Implementace back-end části

U back-end části bude pro implementaci použita platforma Java ve verzi 11.0.2 (JDK 11). Spolu s Javou budeme využívat open-source Spring Boot Framework - verzi 2.3.4.RELEASE - a Gradle nástroj pro automatizované sestavení programu - verzi 6.6.1.

7.1 Spring Framework

Jelikož se jedná o velmi rozsáhlý framework, zmíníme jen jeho hlavní aspekty, a dále se budeme věnovat pouze částem, které byly použity v implementaci. Spring Framework vznikl pro usnadnění vývoje enterprise aplikací. Dle [33] můžeme uvést tyto body:

- Za pomocí návrhového vzoru *Inversion of Control* řeší problém těsných programových vazeb jednotlivých POJO objektů a vrstev.
- Podpora implementace komponent pro přístup k datům, např. formou přímého JDBC či ORM (objektově relačního mapování) nebo nástrojů jako Hibernate.
- Odstranění závislosti na konfiguracích.
- Abstrakce vedoucí ke zjednodušenému používání dalších části J2EE, jako například JMS, JMX, JavaMail, JDBC nebo JNDI.
- Usnadnění psaní unit testů.
- Správa a konfigurační management pro byznys komponenty.

Dalším důležitým vzorem na kterém je Spring postaven je vzor *Dependency Injection*. Ten v tomto případě přesunuje zodpovědnost za vytvoření a provázání závislostí mezi objekty na framework. Objekty jsou tvořeny jako Java Beans.

Kromě samotného Spring Frameworku je nutné také zmínit **Spring Boot**. Spring Boot tvoří nástavbu nad Spring Frameworkem. Poskytuje předdefinované konfigurace a nastavení pro Spring knihovny, díky kterým lze vytvořit velmi rychle funkční aplikaci. Velkou výhodou je, že Spring Boot obsahuje vlastní Tomcat aplikační server. Výslednou aplikaci tak lze jednoduše spustit jako Java proces. Tedy např. pomocí příkazu:

```
java -jar /spring-boot-application.jar
```

7.1.1 Spring Web

Budeme potřebovat vytvořit API pro front-end. K tomu využijeme modul *Spring Web*, který nabízí podpůrné metody pro tvorbu webových rozhraní pomocí Java anotací. Ve výpisu 7.1 můžeme vidět část kódu třídy MainController, kterým definujeme hlavní API rozhraní („endpoint“). (Tělo metody a některé argumenty jsou vynechány pro přehlednost.)

```
1 @GetMapping("/{datasetName}")
2 public void getImageFromBigDataServer(
3     @PathVariable("datasetName") String datasetName,
4     @RequestParam(name = "translate", required = false) String translateParams,
5     HttpServletResponse response) throws IOException {
6
7     // tělo metody
8 }
```

Výpis 7.1: Definice API za pomocí Spring Frameworku

Anotace *GetMapping* určuje metodu pro zpracování HTTP GET požadavku s danou URL cestou. URL cesta bude obsahovat jméno datasetu, které si namapujeme do proměnné *datasetName* typu *String*, za pomocí anotace *PathVariable*. URL bude také obsahovat povinné a nepovinné parametry. Ty získáme a namapujeme do Java proměnných pomocí anotace *RequestParam*. Anotace *RequestParam* obsahuje další atributy, které určují povinnost či nepovinnost parametru, výchozí hodnotu nebo např. pojmenování atributu v URL adrese. Posledním atributem v ukázce je *response* typu *HttpServletResponse*. Jedná se o speciální objekt, který nám dává přístup k objektu HTTP odpovědi. Můžeme pak např. zapisovat binární data do jeho *OutputStream*, přidávat HTTP hlavičky, atd. Ukázali jsme si tedy, jak lze poměrně přehledně definovat webové rozhraní za pomocí Spring Frameworku.

Poslední věcí, kterou v této podkapitole zmíníme je definování a vkládání závislosti ve Spring Frameworku. Zcela jistě budeme potřebovat definovat několik tříd, které budou řešit byznys logiku aplikace, a také budou mezi sebou provázaný. Definice takových objektů spočívá v tvorbě Java rozhraní. Následně se vytvoří třída implementující toto rozhraní s anotací *Service*. Spring mimo jiné používá návrhový vzor *Singleton*, takže v aplikaci bude existovat vždy pouze jedna instance objektu implementující toto rozhraní. Vytvořenou instanci pak můžeme vkládat do ostatních tříd

pomocí anotace *Autowired*. Ve výpisu 7.2 můžeme na příkladu vidět práci se závislostmi ve Spring Frameworku.

```
1  public interface BigDataServerService {
2      // definice metod
3  }
4
5  @Service
6  public class BigDataServerServiceImpl implements BigDataServerService {
7      // implementace rozhraní
8  }
9
10 @RestController
11 public class MainController {
12     private final BigDataServerService bigDataServerService;
13
14     // vložení závislosti
15     @Autowired
16     public MainController(BigDataServerService bigDataServerService) {
17         this.bigDataServerService = bigDataServerService;
18     }
19     [...]
20 }
```

Výpis 7.2: Vytváření a vkládání závislostí pomocí Spring Frameworku

7.2 Struktura back-end aplikace

Samotnou back-end aplikaci lze rozdělit do několika logických částí:

- Aplikační logika
- API
- Konfigurace
- Cache
- Knihovny

Každé logické častí bude věnována jedna kapitola, kde se jí budeme zabývat detailněji. V podstatě lze říct, že každé části odpovídá jeden jmenný prostor, tedy Java balíček (*package*).

7.3 Aplikační logika

V této části je umístěna nejdůležitější a i nejrozsáhlejší část implementace. Nachází se v balíčku cz.kuc0029.bd.image.processor.services. Zde můžeme nalézt BigDataServerService rozhraní,

ImageDataProcessor rozhraní a jejich implementační třídy BigDataServerServiceImpl, resp. ImageDataProcessorImpl. Na obrázcích 7.1 a 7.2 jsou znázorněny jejich třídní diagramy.

7.3.1 BigDataServerService rozhraní

Jedná se o rozhraní, které poskytuje funkcionalitu pro stahování prvotních obrazových dat z BigDataServeru, jejich následnou inicializační transformaci a uložení na souborový systém jako 3D obrázek ve formátu TIF. Třídní diagram můžeme vidět na obrázku 7.1. Pro konverzi stažených (binárních) obrazových dat do 3D obrázku se využívá knihovna ImgLib2. Rozhraní také zároveň poskytuje přístup k metadatům daného datasetu - metoda getDatasetMetadata(...). Čtení metadat je využíváno ještě před samotným stahováním obrazových dat, protože je potřeba zjistit počet dimenzí daného datasetu a ověřit, zda-li je požadovaná konfigurace (tj. kombinace *timepoint, setup, level*) dostupná. Stahování samotných obrazových dat je prováděno vícevláknově za pomocí třídy java.util.concurrent.Executors. Počet vláken je konfigurovatelný pomocí třídní proměnné *FETCH_CELLS_DATA_THREADS*.

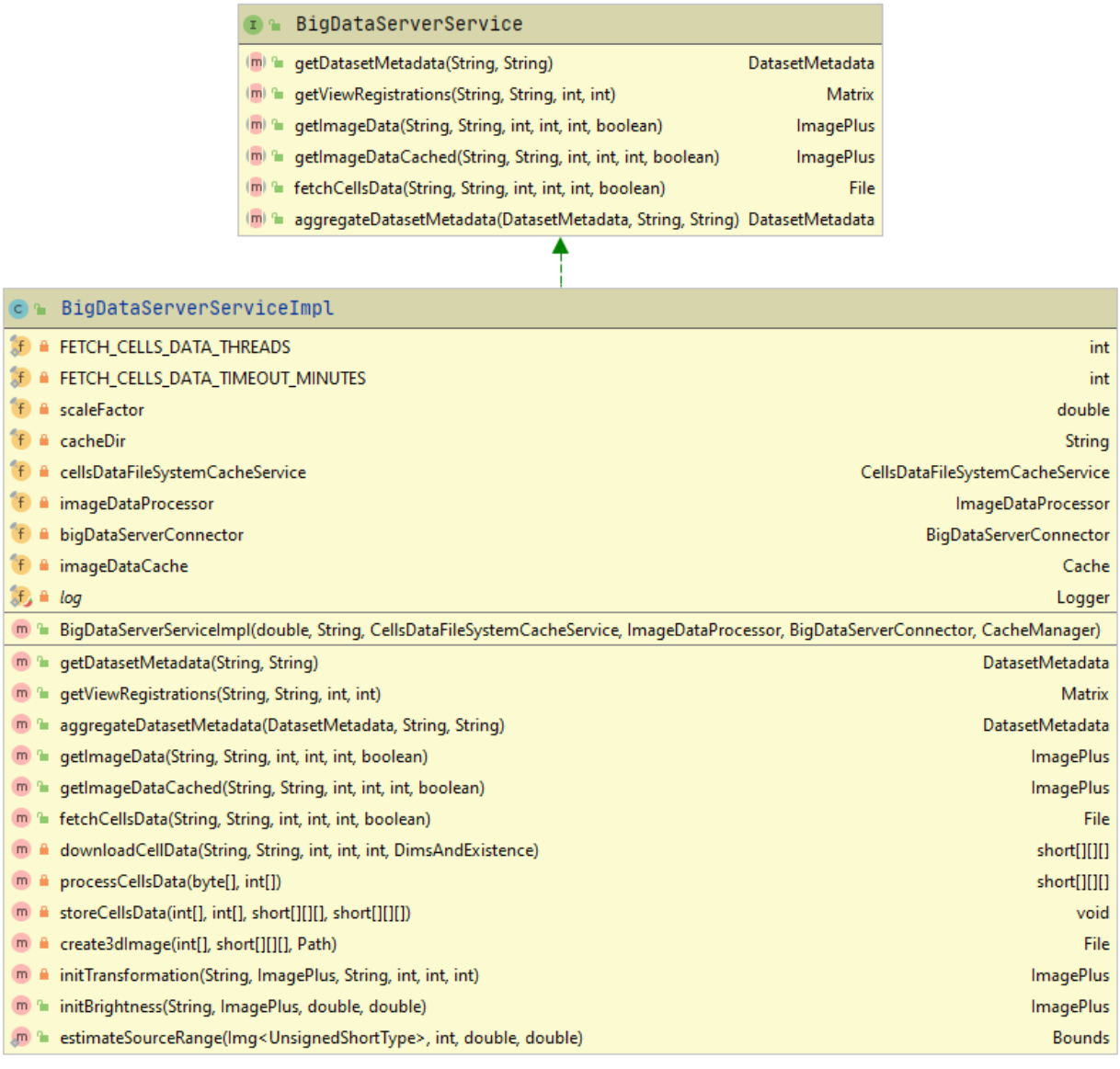
Po stažení a konverzi dat na 3D obrázek je potřeba provést inicializační transformaci dle XML souboru datasetu. Transformační matici získáme zavoláním metody getViewRegistrations(...). Samotnou transformaci provádí na pozadí knihovna TransformJ. Součástí inicializační transformací je i korekce jasu obrazových dat - metoda initBrightness(...). Výsledný transformovaný obrázek je pak znovu uložen na souborový systém a vrácen jako návratová hodnota metody getImageData(...). Poznamenejme, že v tomto místě už pracujeme s obrázky jako s objekty třídy ij.ImagePlus. Instanci ImagePlus reprezentující uložený obrázek ze souborového systému získáme zavoláním konstruktoru a jako parametr mu předáme absolutní cestu k souboru s obrázkem.

U inicializačních operací ještě zůstaneme a zmíníme proměnou scaleFactor. Její hodnota je konfigurovatelná a zásadním způsobem ovlivňuje rychlost a paměťovou náročnost transformace. Scale factor je desetinné číslo, kterým se násobí všechny prvky výchozí transformační matice z XML souboru datasetu. Tuto operaci budeme nazývat škálování transformační matice. Škálování také pochopitelně ovlivňuje vlastnosti výsledného inicializovaného obrázku jako šířku, výšku, hloubku, velikost atd. Při vývoji aplikace byla pro testovací datasety nastavena hodnota scaleFactor na 0.46 jako kompromis mezi rychlostí a kvalitou obrázků.

7.3.2 ImageDataProcessor rozhraní

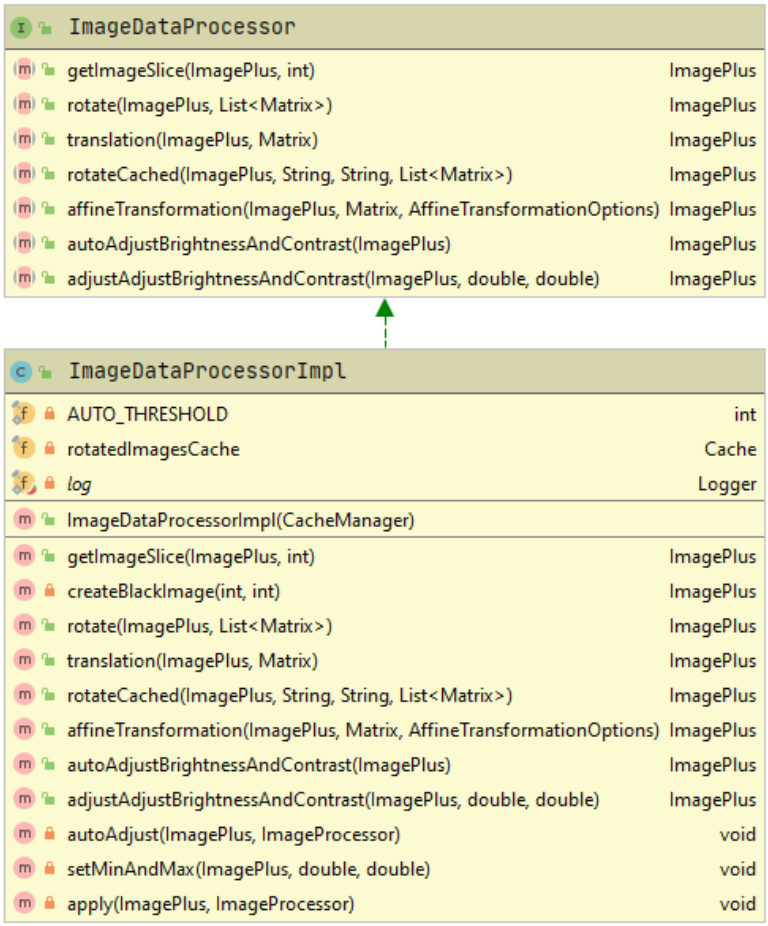
Toto rozhraní poskytuje metody pro práci s obrázky jako je rotace, translace, úprava jasu nebo získání konkrétního řezu. Transformace jsou na pozadí prováděny knihovnou TransformJ [34].

Metoda getImageSlice(...) vrací požadovaný řez z daného 3D obrázku. Řezy jsou indexovány od nuly. Pracuje se zde s objekty typu ImagePlus. Za zmínku stojí také to, že pokud uživatel požaduje řez s indexem mimo hranice obrázku, je vygenerován prázdný 8-bitový černý obrázek - metoda createBlackImage(...).



Obrázek 7.1: Třídní diagram - *BigDataServerService*

Všechny operace nad obrázky se provádí jako afinní transformace (viz kapitola 4.7). Metody rotate(...) a translation(...) tedy přijímají na svém vstupu transformační matice typu cz.kuc0029.bd.image.processor.util.Matrix. Tyto transformační matice jsou vypočteny na základě vstupu od uživatele z front-end komponenty a následně mapovány na vstupní parametry pro TransformJ knihovnu. Knihovnu TransformJ si popíšeme v kapitole 7.8.



Obrázek 7.2: Třídní diagram - *ImageDataProcessor*

7.4 API

Webové rozhraní tvoří dva API přístupové body. Tabulky 7.1 a 7.2 popisují jejich parametry.

- `/{datasetName}` - vrací (transformované) obrázky ve formátu PNG dle parametrů.

Tabulka 7.1: Popis API parametrů pro koncový bod `/{datasetName}`

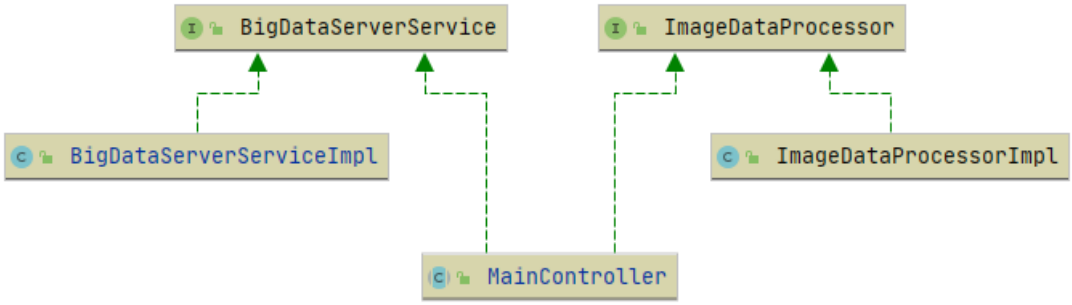
Parametr	Popis	Povinný	Výchozí hodnota
datasetName	Jméno datasetu. Jméno musí korespondovat s konfigurací BigDataServeru. *Parametr je součásti URL cesty.*	Ano	
uniqueFeId	Unikátní identifikátor relace (session id).	Ano	
bigDataServerUrl	URL adresa BigDataServeru, např. `http://julius2.it4i.cz`.	Ano	
timepoint	Číselná hodnota timepoint.	Ano	0
setup	Číselná hodnota setup id.	Ano	0
level	Číselná hodnota level.	Ano	2
slice	Číselná hodnota řezu.	Ano	1
translate	Konfigurace translace ve formátu „x:x_0_y:y_0“. Hodnoty x_0 a y_0 představují vzdálenost v pixelech.	Ne	x:0_y:0
rotate	Konfigurace rotace ve formátu „x:x_0_y:y_0“. Hodnoty x_0 a y_0 představují stupně. Např. „x:90“ značí rotaci okolo osy x o 90° po směru hodinových ručiček.	Ne	x:0_y:0
autoAdjust	Pokud je příznak nastaven na hodnotu „true“, zapíná se další úroveň úpravy jasu a kontrastu. Lze použít při nedokonale osvětlených vzorcích, ale může také způsobit přesvětlení vzorku.	Ne	false

- `/metadata/{datasetName}` - přístupový bod vrací metadata o datasetu ve formátu JSON.

Tabulka 7.2: Popis API parametrů pro koncový bod `/metadata/{datasetName}`

Parametr	Popis	Povinný	Výchozí hodnota
datasetName	Jméno datasetu. Jméno musí korespondovat s konfigurací BigDataServeru. *Parametr je součásti URL cesty.*	Ano	
bigDataServerUrl	URL adresa BigDataServeru, např. `http://julius2.it4i.cz`.	Ano	

Definici API můžeme najít v třídě MainController. Následující obrázek 7.3 ukazuje závislost třídy MainController na výše zmíněných rozhraních.



Obrázek 7.3: Závislosti tříd

7.5 Konfigurace

Konfiguraci lze najít v `bd-image-processor/src/main/resources/application.yml` souboru. V produkčním sestavení (release verzi) se tento konfigurační soubor nachází zabalený v JAR souboru. Nicméně konfigurační parametry lze předat i při spuštění jako argumenty. Případně existuje také možnost předat argumentem cestu k externímu konfigurační souboru. Důležitými konfiguračními parametry jsou „server.port“ - určuje port, na kterém bude back-end naslouchat příchozím požadavkům; „cache.dir“ - složka v souborovém systému, kde budou ukládány inicializované obrázky; „transformation.scale-factor“ - scale factor. Zmíněné konfigurační parametry jsou důležité pro spuštění aplikace. Kromě nich lze předávat další konfigurace, které upravují chování Spring Frameworku, viz dokumentace[1].

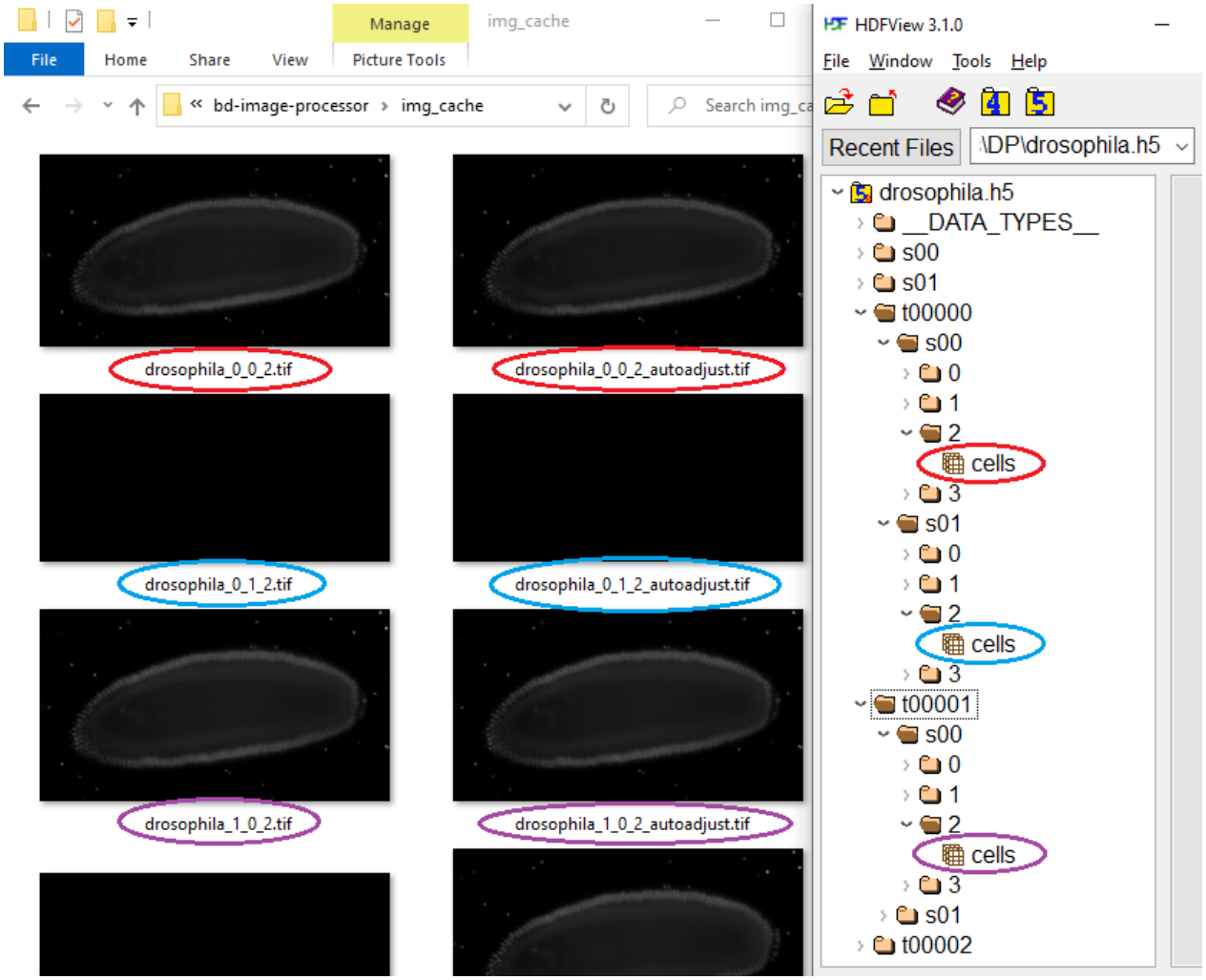
7.6 Cache

Cache tvoří důležitou součást aplikace, která zajišťuje stabilní odezvu při práci s aplikací. V této kapitole si popíšeme typy použité cache a strategie cachování.

Prvním typem je cachování obrázku na souborovém systému hned poté, co jsou staženy obrazová data z BigDataServeru a jsou provedeny inicializační operace. Takto se ukládají inicializované obrázky pro konkrétní dataset, *timepoint, setup* a *level*. Pokud je povolena možnost *autoAdjust* (viz kapitola 7.4), vznikne zvláštní soubor se sufixem „_autoadjust“ pro tuto možnost. Soubory se ukládají do složky, jejíž cesta je specifikována konfiguračním parametrem `cache.dir`. Takto uložené soubory přímo odpovídají datům ve struktuře formátu H5 (obrázek 7.4).

[1] Viz `https://docs.spring.io/spring-framework/docs/5.2.9.RELEASE/spring-framework-reference`

Dalšími dvěma typy cache jsou paměťové cache, jenž se používají pro cachování mezivýsledků při transformačních operacích. Jejich konfiguraci můžeme nalézt ve třídě CachingConfig. Pro cache využíváme podporu Spring Frameworku, konkrétně Spring *CacheManager*. Jedná se o objekt, který má na starost centrální správu cache a mimo jiné také umožňuje pojmenování cache regionů. Nyní si definujeme konkrétněji tyto dva další typy cache paměti, jejichž pojmenování uchovávají třídní proměnné IMAGES_DATA_CACHE a ROTATED_IMAGES_CACHE v třídě CachingConfig.



Obrázek 7.4: bd-image-processor - cachované obrázky korespondují s daty ve struktuře H5. Názvy souborů vlevo odpovídají formátu *dataset_timepoint_setup_level[_autoadjust].tif*.

Jelikož přístup k disku je řádově pomalejší něž přístup k paměti (i v případě modernějších SSD disků), je potřeba právě prohlížené obrázky uchovávat v cache paměti.
IMAGES_DATA_CACHE slouží právě pro tyto účely. Uložené obrázky jsou uchovány v cache jen po omezenou dobu, pokud nejsou aktivně používaný. Tato cache má signifikantní význam při prohlížení řezů a translaci. Díky ní je prohlížení řezů plynulé. Do této cache se ukládají objekty

typu ImageCacheEntry. Ty obsahují kromě samotných obrazových dat i unikátní cache klíč, pod kterým je cachovaný záznam uložen a časový údaj o posledním použití. Proměnná s časovým údajem o posledním použití se aktualizuje při každém přečtení dat. Pokud data nejsou po nějakou dobu čtena, jsou z cache odstraněna. Konkrétní hraniční časová hodnota (TTL) je uvedena v třídě CacheGarbageCollector.

ROTATED_IMAGES_CACHE je principiálně totožná s IMAGES_DATA_CACHE. Rozdíl je v tom, že se do ní ukládají obrázky, které byly nějakým způsobem rotovány. Protože rotování a translace se provádí vždy od znovu a za běhu nad stejnými vstupními daty, je výhodné také cachovat aktuálně rotovaný obrázek pro možnost prohlížení řezů při dané rotaci. Rotace bývají časově náročnější operace a bez této cache by každá změna řezu již rotovaného obrázku vyžadovala provedení rotace anebo translace znovu a nebyla by zachována plynulost.

7.7 Knihovny

V této sekci budou probírány knihovny použité v back-end části. Jsou to knihovny ImgLib2 [21, 4] a TransformJ [34] pro podporu práce s obrázky. Také si zde ukážeme části kódu, kde jsou knihovny využity.

7.7.1 ImgLib2 - ImageJ

V kapitole 4.4 jsem se již zabývali knihovnou ImgLib2. Popsali jsem si její principy, cíle a hlavní funkcionalitu co se týče Fiji BigDataVieweru. V této sekci si popíšeme aspekty knihovny z programátorského hlediska.

Obrázek je tímto frameworkem definován jako libovolné mapování z podmnožiny n-dimenzionálního euklidovského prostoru do hodnoty obecného typu „pixel“. ImgLib2 knihovna je postavena na třech hlavních konceptech (*zachováme anglické názvy*): accessibles, accessors a types [4].

Accessibles

Obrázky jsou v ImgLib2 reprezentovány jako *accessibles*. *Accessibles* reprezentují data jako taková, např. rastrový obrázek. Všechny třídy pro práci s pixely jsou typu *RandomAccessible*. *RandomAccessible* a *RandomAccessibleInterval* rozhraní poskytují samotný přístup k pixelu dané souřadnice. Kromě těchto dvou rozhraní zmíníme ještě *IterableInterval*. *RandomAccessibleInterval* a *IterableInterval* reprezentují tzv. ohraničené obrázky, kde jsou všechny pixely umístěny v určitém intervalu. Zjednodušený UML diagram, reprezentující *accessibles*, je v příloze A, obrázek A.2. *Accessibles* nám dává přístup k *accessor*.

Accessors

Pomocí *accessors* se v ImgLib2 manipuluje s obrázky. Pokud budeme mít obrázek složený z pixelů, *accessor* bude představovat (pohyblivou) referenci na pixel. S takovou referencí (*accessorem*) můžeme pohybovat po obrázku a vybírat náhodně pixely; můžeme udělat tzv. dereferenci a získat hodnotu vybraného pixelu anebo získat souřadnice pixelu na který reference ukazuje. ImgLib2 nabízí dva základní typy *accessors*:

- *RandomAccess* - poskytuje náhodný n-dimenzionální přístup přes rozhraní *Positionable* a může být nastaven na libovolnou souřadnici.
- *Cursor* - umožňuje iteraci přes klasické Java rozhraní *Iterator*. Může být posouván vpřed a iterovat tak všemi pixely. Důležité je uvědomit si, že pořadí iterace není z důvodů optimalizace definováno.

Oba tyto typy implementují rozhraní *Sampler* pomocí něhož lze získat hodnotu pixelu. Dále implementují *Localizable* rozhraní, které dovoluje získat souřadnice aktuálního pixelu a *EuclideanSpace* rozhraní, což dovoluje získat počet dimenzí obrázku. Poznamenejme, že *Sampler, RandomAccess* a *Cursor* mají <T> parametr, který značí v Javě generický datový typ.

Cursor obecně dosahuje výrazně lepších rychlostí než *RandomAccess*. Zjednodušený UML diagram *accessors* můžeme opět vidět v příloze A, obrázek A.1. *Accessors* nám dávají přistup k *types*.

Types

ImgLib2 zahrnuje běžné datové typy pro práci s obrazovými daty jako *BitType*, *UnsignedShortType*, *ARGBType* apod. a operace nad nimi. Tyto datové typy jsou implementovány jako proxy objekty na Java primitivní typy (*byte[], float[]*). Seznam všech podporovaných typů můžeme nalézt v dokumentaci. Co se týče kolekcí, ImgLib2 nabízí např. *ListImg* nebo *ArrayImg*. *ListImg* ukládá pixely jako individuální instance a podporuje volitelný datový typ, který dědí z *net.imglib2.type.Type*, nebo také obecný Java Object. *ArrayImg* mapuje datové typy z ImgLib2 do datového typu pole a poskytuje tak optimální rychlost. Nutno však podotknout, že Java pole jsou limitovány velikostí 2^{31} px. To nám umožňuje např. uložit 2D čtvercový obrázek s maximální velikostí strany 46 340 px. Takové omezení je pro nás dost limitující a může být lehce překročeno. V takovém případě musíme použít *CellImg* typ. Ten rozděluje souřadnice do n-dimenzionální mřížky buněk. Každá taková buňka je mapována na primitivní typ pole. *CellImg* umožňuje ukládat podstatně větší obrázky - až 2^{62} px - za cenu lehce sníženého výpočetního výkonu [4].

Na ukázce níže můžeme vidět metodu create3dImage(...), jejíž cílem je vytvořit z binárních obrazových dat z BigDataServeru obrázek ve formátu TIF. Jako vstupní parametry metody předáváme počet dimenzí obrázku (šířka, výška, hloubka), tj. proměnná *int[] dimensions*. Jako druhý vstupní parametr předáváme obrazová data jako taková ve trojrozměrném poli typu *short*. Posledním parametrem je absolutní cesta na souborovém systému pro uložení obrázku. Cesta zároveň obsahuje

název souboru a příponu. Podle přípony se uvnitř knihovny ImgLib2 volí konverze dat. Je proto důležité ověřit si, jestli je používaný formát podporovaný knihovnou. V ukázce můžeme také vidět použití *Cursor* rozhraní pro nastavení pixelu na správnou souřadnici výsledného obrázku. Obrazová data jsou v poli *imageData* uspořádána tak, jako ve výsledném obrázku. Nakonec pro uložení obrázku využijeme *ImgSaver* a vrátíme objekt *java.io.File*. Ukázku můžeme vidět ve výpisu 7.3.

```
1  private File create3dImage(int[] dimensions,
2                             short[][][] imageData,
3                             Path filePath) {
4
5      final Img<UnsignedShortType> img = new ArrayImgFactory<>(
6          new UnsignedShortType()).create(dimensions);
7
8      Cursor<UnsignedShortType> cursor = img.localizingCursor();
9      while (cursor.hasNext()) {
10         cursor.fwd();
11
12         int w = cursor.getIntPosition(0);
13         int h = cursor.getIntPosition(1);
14         int z = cursor.getIntPosition(2);
15
16         cursor.get().set(imageData[z][w][h]);
17     }
18     File imagej2 = new File(filePath.toUri());
19     ImgSaver imgSaver = new ImgSaver();
20     imgSaver.saveImg(imagej2.getAbsolutePath(), img);
21     imgSaver.context().dispose();
22     return imagej2;
23 }
```

Výpis 7.3: Ukázka kódu - knihovna ImgLib2

7.8 TransformJ

TransformJ je balíček od ImageJ pro geometrické transformace a manipulace s obrázky. Tato knihovna umožňuje zpracovávat obrázky s maximálně pěti dimenzemi. Lze ji nainstalovat i jako plugin do Fiji ImageJ distribuce. Kompletní výpis podporovaných operací TransformJ můžeme nalézt v [34]. Nás bude zajímat pouze sekce „Affine“, tedy afinní transformace nad obrázky.

Pro afinní transformace požaduje knihovna na svém vstupu několik parametrů (v závorce jsou uvedeny korespondující anglické názvy z dokumentace):

1. Matice (Matrix).
2. Interpolační schéma (Interpolation).

3. Pozadí (Background).
4. Úprava hranic obrázku (Adjust bounds to fit result).
5. Izotropní vzorkování (Resample isotropically).
6. Oprava zubatosti (Anti-alias borders).

Základní parametr je matice 4x4, jenž popisuje danou transformaci. Při sestavování matice je předpoklad použití pravidla pravé ruky pro souřadnicový systém. To znamená, že osa x je dána vodorovně, osa y svisle a osa z je dána kolmo k pozorovateli. Pokud budeme k souřadnici x přičítat pohybujeme se zleva doprava. V případě přičítání k souřadnici osy y, se pohybujeme shora dolů. Při přičítání k souřadnici z se na ose z budeme pohybovat dále od pozorovatele.

Dalším atributem je výběr interpolačního schématu. TransformJ nabízí různé typy interpolačních schémat, viz [26]. My budeme využívat lineární interpolační schéma. Třetím parametrem je barva pozadí pro části obrázku, které se stanou vlivem transformací nevyplněny. V našem případě bude použita černá barva pro vyplnění.
Afinní transformace můžou způsobit, že se část nebo celý obrázek přesune za hranice původního obrázku. Tím se dostáváme k čtvrtému parametru. Povolením možnosti „Adjust bounds to fit result“ umožníme TransformJ upravit hranice výsledného obrázku tak, aby se výsledný transformovaný obrázek vlezl do těchto hranic.
Poslední dva atributy „Resample isotropically“ a „Anti-alias borders“ mají vliv na výslednou kvalitu obrázku po transformaci. Ve výchozím nastavení bude mít výsledný obrázek stejnou velikost voxelů jako vstupní obrázek. V případě anizotropních voxelů to může mít za následek ztrátu informace. Například při rotaci se může stát rovina x-y s velkým rozlišením rovinou x-z s nízkým rozlišením [26]. Při výběru možnosti „Resample isotropically“ je výsledný obrázek při transformacích převzorkován a jsou získány izotropní voxely, které se rovnají nejmenším voxelům šířky, výšky a hloubky vstupního obrázku. Na druhou stranu to vyžaduje více času pro transformaci. „Anti-alias borders“ možnost zajišťuje hladší přechody mezi hranicemi transformovaného obrázku a barvou pozadí.

Část kódu ve výpisu 7.4 zobrazuje metodu, která provádí na back-endu afinní transformace. Význam parametrů jsme si popsali výše. Poznamenejme, že parametry *adjust, resample a antialias* jsou povoleny (tj. nastaveny na hodnotu true) jen při inicializačních transformacích. Při transformacích prováděny uživatelem se používá následující nastavení: *adjust* vypnuto (false), *resample a antialias* povoleno (true).

```
public ImagePlus affineTransformation(ImagePlus image,
    Matrix affineMatrix, AffineTransformationOptions options) {

    Calibration calibration = image.getCalibration();
    final int interpolation = options.getInterpolation();
    final double bg = options.getBackground();
    final boolean adjust = options.isAdjust();
    final boolean resample = options.isResample();
    final boolean antialias = options.isAntialias();

    final int scheme;
    switch (interpolation) {
    // výběr interpolačního schématu, výchozí nastavení používá Affine.LINEAR
    }

    final Transform transform = new Transform(affineMatrix.getData());
    final Image input = Image.wrap(image);
    final Affine affiner = new Affine();
    affiner.background = bg;

    final Image output = affiner.run(input, transform, scheme, adjust,
        resample, antialias);
    ImagePlus outputImagePlus = output.imageplus();
    outputImagePlus.setCalibration(calibration);
    return outputImagePlus;
}
```

Výpis 7.4: Ukázka kódu - afinní transformace za pomocí knihovny TransformJ

7.9 První spuštění

Pro spuštění aplikace bd-image-processor je potřeba nejdříve nainstalovat všechny závislosti. Instalování závislostí je provedeno automatizovaně pomocí nástroje Gradle. Postup je následující:

1. Přepneme se do složky `bd-image-processor`.
2. Pokud instalaci provádíme na Windows, zadáme příkaz: „gradlew.bat build“.
   Pro UNIX OS zadáme příkaz: „./gradlew build“.
   Poznámka: je nutné mít nainstalovanou správnou verzi Javy (kapitola 7) a správně nastavenou systémovou proměnnou JAVA_HOME.
3. Po úspěšném sestavení nalezneme ve složce `bd-image-processor/build/libs` JAR soubor. Název souboru bude obsahovat aktuální verzi. Přepneme se tedy do složky `build/libs`.
4. Nyní je potřeba vytvořit složku, kde budou ukládány stažené a inicializované obrázky. My si vytvoříme ve složce `build/libs` složku `img_cache`

5. Aplikaci můžeme následně spustit a přes parametr předat cestu k vytvořené složce. V našem případě použijeme příkaz:

   ```
   java -jar bd-image-processor.jar -DCACHE_DIR=./img_cache
   ```

6. Po nastartování aplikace se spustí HTTP server, který bude naslouchat na nastaveném portu. Výchozí port je 8081.

Kapitola 8

Zhodnocení webové aplikace

V této kapitole zhodnotíme vytvořenou aplikaci, shrneme její přednosti i nedostatky a nastíníme jakým směrem by se mohl odebírat její budoucí vývoj.

8.1 Vlastnosti webové aplikace

Vytvořená aplikace je funkční a umožňuje vizualizovat data poskytovaná libovolným BigDataServerem. Co se týče aktuálně implementované funkcionality a ovládaní, řešení vychází z Fiji BigDataViewer pluginu. Aplikace umožňuje při prohlížení vybírat mezi více zdroji dat, tj. *setups* (pokud jsou dostupná), upravovat jas a kontrast zobrazení, procházet jednotlivé *timepoints*, provádět rotaci a translaci nebo prohlížet jednotlivé řezy pozorovaného vzorku. Více ukázek v podobě screenshots můžeme nalézt v příloze B.

Mezi přednosti této implementace bych zařadil hlavně technologie. Pro implementaci byly použity aktuální technologie, které nabízí spoustu možností dalšího vývoje a použití. Také není problémové nasadit vytvořenou aplikaci kdekoliv, ať už na vzdálený server nebo do cloudového prostředí, protože pro zmíněné technologie existuje mnoho předpřipravených řešení, které umožňují instantně spustit kontejner s Java nebo React aplikací. Jako příklad můžeme uvést RedHat Openshift prostředí, což je softwarové řešení postavené na Docker kontejnerech. Openshift umožňuje mimo jiné vytvářet a nasazovat Docker kontejnery ze zdrojového Git repozitáře.

8.2 Výhody

Výhodou React technologie je také její kompatibilita napříč většinou aktuálních webových prohlížečů na trhu. Samotná implementace je navržena tak, že veškeré výpočetně náročné operace řeší back-end část běžící na serveru. Front-end část řeší pouze „jednoduché“ operace jako zobrazovaní obrázků a ovládání. To znamená, že výkon aplikace není do jisté míry degradován prostředím, ve kterém prohlížeč běží. Jedním ze zadaných cílů byla škálovatelnost aplikace. Implementované řešení je

škálovatelné a to jak horizontálně, tak vertikálně. Horizontální škálování je samozřejmě možné jen do určité míry a má své limity. Vertikální škálování teoreticky nějak omezené není. Je možné např. nasadit více instancí back-end aplikace v cloudovém prostředí a přes aplikační *load balancer* rozdělovat mezi nimi zátěž.

8.3 Nedostatky

Mezi hlavní nedostatky této verze aplikace patří její doba odezvy při rotačních operacích a to v závislosti na vlastnostech daného datasetu (i přes použití cachování). Konkrétně v závislosti na rozlišení obrazových dat v datasetu. Pro určité testovací datasety je doba odezvy velmi dobrá (pod 2 sekundy), pro jiné datasety šplhá doba odezvy pro rotační operace až k trojnásobku běžné doby (okolo 6 sekund). To může být pro uživatele nepříjemné. Důvodem může být provádění rotačních operací za běhu knihovnou TransformJ, tedy její implementace, kterou může být potřeba více optimalizovat. Ostatní operace jako translace nebo změna řezu těmito nedostatky netrpí (doba odezvy okolo 1 sekundy). Aktuální verze aplikace neobsahuje žádný mechanismus pro správu cachovaných obrázků na souborovém systému (konfigurační parametr „cache.dir“). Takový mechanismus by se hodil, protože by eliminoval potřebnou správu aplikace ve smyslu promazávání složky s cachovanými obrázky. Další drobnou výtkou může být design, jenž by si zasloužily jistě lepší vzhled, nicméně design nebyl primárním cílem při implementaci, protože se jedná o nefunkční požadavek. Také je potřeba lépe vyřešit zobrazování chybových stavů v aplikaci.

Ještě dodejme, že aplikace byla vyvíjena a testována na notebooku Dell Latitude 5590[1] a byly použity testovací datasety „Drosophila“, „HisYFP-SPIM“ na buďto lokálně spuštěnému BigDataServeru, nebo BigDataServeru na adrese `http://julius2.it4i.cz`. Primárním prohlížečem pro testování aplikace byl prohlížeč Google Chrome (verze 89.0.4389.90, 64-bit), ale byly také úspěšné odzkoušeny prohlížeče Firefox (verze 85.0.2, 32-bit) a Microsoft Edge (verze 89.0.774.57, 64-bit).

8.4 Budoucí vývoj

Ze zmíněných nedostatků vyplývá, že další vývoj by se měl zabývat mimo jiné optimalizací aplikace. Především optimalizací provádění rotačních operací. Tzn. zaměřit se na knihovnu TransformJ a podívat se detailněji na její algoritmy, zdali je možná nějaká optimalizace pro dosažení stabilnější odezvy pro rotační operace. Optimalizace algoritmu by se také hodila pro inicializační operace, konkrétně metody pro vytváření 3D obrazových dat. Větší rozlišení obrazových dat se podepisuje delší dobou provádění této metody. Jedna z možností je zkusit použít vícevláknový přístup. Drobné zlepšení může také přinést volba ztrátového formátu pro posílané obrázky mezi back-end a front-end

[1] Procesor Intel(R) Core(TM) i7-8650U 1.90GHz, paměť 32 GB, disk SSD 475GB, OS Windows 10 Enterprise.

komponentami. Ztrátový formát bude znamenat menší objem dat potřebných pro přenos, ale na úkor kvalitě obrazových dat.

Kromě optimalizace aplikace by mohl další vývoj přinést funkce, jež jsou v BigDataVieweru a v aplikací chybí (např. „fused mode“), ovládání pomocí klávesových zkratek, centrování pohledů atp.

Co se týče vzhledu aplikace, tedy front-end části, lze zapracovat zcela jistě na modernější vzhledu aplikace a udělat stránky plně responsivní.

Kapitola 9

Závěr

Cílem této práce je navržení a implementace webové komponenty pro zobrazování obrazových dat BigDataServeru. Výsledkem je funkční aplikace, jenž se skládá ze dvou komponent a umožňuje zobrazovat a provádět transformace s obrazovými daty poskytovanými BigDataServerem. Implementace vychází z pluginu BigDataViewer desktopové aplikace Fiji ImageJ a byly pro ni použity aktuální technologie React a Spring Framework.

Před samotnou implementací bylo potřeba nastudovat a popsat způsob vizualizace BigDataVieweru, BigDataServer a jeho webové služby i samotný formát XML/HDF5, který se používá pro ukládání datasetů. BigDataViewer i výsledná webová aplikace je postavena na afinních transformacích ve 3D prostoru. Tomuto tématu jsme věnovali několik sekcí této práce a ukázali jsme si je i na příkladu. Pro práci s obrázky a obrazovými daty se využívá knihovna ImgLib2, jenž poskytuje pokročilejší a efektivnější práci s obrázky. Byl diskutován také renderovací algoritmus BigDataVieweru.

V kapitole popisující BigDataServer jsme diskutovali, jaké technologie používá na pozadí a popsali jsme jeho poskytované webové služby. Uvedli jsme potřebné parametry pro spuštění a jak se lze napojit na konkrétní BigDataServer z Fiji ImageJ aplikace. Klíčové bylo také pochopení formátu HDF5, respektive jeho modifikace XML/HDF5, kterou využívá samotný BigDataServer a je optimalizována pro rychlý přístup k velkým datovým sadám v rámci terabajtů. HDF5 technologii i XML/HDF5 formát jsme popsali přímo na reálném datasetu. Součástí kapitoly o BigDataServeru jsou i Mipmap Pyramidy. U těch jsme si popsali, jak jsou v HDF5 souboru uloženy a jakými principy přispívají k optimalizaci.

Následně popis praktické části této práce tvoří tři kapitoly v kterých jsme se detailně věnovali návrhu aplikace, implementaci front-end a back-end části. Specifikovali jsme si cíle aplikace a navrhli hrubý náhled na komponenty aplikace (tzv. „*high-level overview*“). Dále jsme uvedli dané technologie React a Spring framework do kontextu dané problematiky, nastínili řešení v daných technologiích, navrhli komponenty a webové API. Popsali jsme také použité pomocné knihovny

i důvod proč jsme je zvolili. Součástí kapitol praktické části je také postup na první spuštění, popis navigace a ovládání webové aplikace.

Zhodnocení výsledné aplikace jsme popsali již v kapitole 8. Uvedeme jen, že webová aplikace umožňuje napojení na libovolný BigDataServer, zobrazení obrazových dat včetně volby různých *views* a základní operace s obrazovými daty jako rotace, translace, úprava jasu apod. Není už tedy potřeba instalovat desktopovou aplikaci Fiji, ale k vizualizace lze použít webový prohlížeč. Tímto se otevírají nové možnosti pro vizualizování dat. Zmínili jsme, že další vývoj by se měl zabývat především optimalizací, protože pro některé reálné datasety dochází k větším časovým prodlevám co se týče odezvy při práci s aplikací. Samotná aplikace, respektive její front-end část, je pak zachycena v příloze B na obrázku B.1. Případně její využití jako plugin vložený do jiné stránky na obrázcích B.2, B.3.

Literatura

1. HUISKEN, Jan; SWOGER, Jim; BENE, Filippo Del; WITTBRODT, Joachim; STELZER, Ernst H K. Optical Sectioning Deep Inside Live Embryos by Selective Plane Illumination Microscopy. *Science*. 2004-08-13, roč. 305, č. 5686, s. 1007–1009. ISSN 0036-8075. Dostupné z DOI: 10.1126/science.1100035.
2. HUSZ, Zsolt L.; BURTON, Nicholas; HILL, Bill; MILYAEV, Nestor; BALDOCK, Richard A. Web tools for large-scale 3D biological images and atlases. *BMC Bioinformatics*. 2012-06, roč. 13, s. 122. ISSN 1471-2105. Dostupné z DOI: 10.1186/1471-2105-13-122.
3. SCHINDELIN, Johannes; ARGANDA-CARRERAS, Ignacio; FRISE, Erwin; KAYNIG, Verena; LONGAIR, Mark; PIETZSCH, Tobias; PREIBISCH, Stephan; RUEDEN, Curtis; SAALFELD, Stephan; SCHMID, Benjamin; TINEVEZ, Jean-Yves; WHITE, Daniel; HARTENSTEIN, Volker; ELICEIRI, Kevin; TOMANCAK, Pavel; CARDONA, Albert. Fiji: An Open-Source Platform for Biological-Image Analysis. *Nature methods*. 2012-06, roč. 9, s. 676–82. Dostupné z DOI: 10.1038/nmeth.2019.
4. PIETZSCH, Tobias; PREIBISCH, Stephan; TOMANČÁK, Pavel; SAALFELD, Stephan. ImgLib2—generic image processing in Java. *Bioinformatics*. 2012-11-15, roč. 28, č. 22, s. 3009–3011. ISSN 1460-2059. Dostupné z DOI: 10.1093/bioinformatics/bts543.
5. O' DONOGHUE, Sean; GAVIN, Anne-Claude; GEHLENBORG, Nils; GOODSELL, David; HÉRICHÉ, Jean-Karim; NIELSEN, Cydney; NORTH, Chris; OLSON, Arthur; PROCTER, Jim; SHATTUCK, David; WALTER, Thomas; WONG, Bang. Visualizing biological data-now and in the future. *Nature methods*. 2010-03, roč. 7, S2–4. Dostupné z DOI: 10.1038/nmeth.f.301.
6. PROCTER, Jim; THOMPSON, Julie; LETUNIC, Ivica; CREEVEY, Chris; JOSSINET, Fabrice; BARTON, Geoffrey. Visualization of multiple alignments, phylogenies and gene family evolution. *Nature methods*. 2010-03, roč. 7, S16–25. Dostupné z DOI: 10.1038/nmeth.1434.
7. O' DONOGHUE, Sean; GOODSELL, David; FRANGAKIS, Achilleas; JOSSINET, Fabrice; LASKOWSKI, Roman; NILGES, Michael; SAIBIL, Helen; SCHAFFERHANS, Andrea; WADE, Rebecca; WESTHOF, Eric; OLSON, Arthur. Visualization of Macromolecular Structures. *Nature methods*. 2010-03, roč. 7, S42–55. Dostupné z DOI: 10.1038/nmeth.1427.

8. GEHLENBORG, Nils; O' DONOGHUE, Sean; BALIGA, Nitin; GOESMANN, Alexander; HIBBS, Matthew; KOHLBACHER, Oliver; NEUWEGER, Heiko; SCHNEIDER, Reinhard; TENENBAUM, Dan; GAVIN, Anne-Claude. Visualization of omics data for system biology. *Nature methods*. 2010-03, roč. 7, S56–68. Dostupné z DOI: 10.1038/nmeth.1436.

9. DAVID, Martin; JAMES, Procter; BEN, Soares; ANDREW, Waterhouse; SAIF, Shehata; NANCY, Giang; MUNGO, Carstairs; CHARLES, Ofoegbu; KIRA, Mourão; SUZANNE, Duce; GEOFF, Barton. *Jalview 2.11: Manual and Introductory Tutorial* [online]. School of Life Sciences, University of Dundee, Scotland DD1 5EH, UK, 2020-10-07 [cit. 2021-04-07]. Dostupné z: http://www.jalview.org/sites/jalview.org/files/TheJalviewTutorial_0.pdf.

10. HANSEN, C.D.; HANSEN, C.D.; JOHNSON, C.R.; PUBLISHERS, Elsevier Science; INC, Engineering Information. *Visualization Handbook*. Elsevier Science, 2005. ISBN 9780123875822. Dostupné také z: https://books.google.cz/books?id=ZFrlULckWdAC.

11. *VTK - The Visualization Toolkit: VTK in Action* [online] [cit. 2021-04-06]. Dostupné z: https://vtk.org/vtk-in-action/#image-gallery.

12. TOBIAS, PIETZSCH; HONGKEE, MOON; PAVEL, TOMANCAK. *BigDataServer: ImageJ* [online]. 2015-10-08 [cit. 2021-01-13]. Dostupné z: https://imagej.net/BigDataServer.

13. PIETZSCH, Tobias; SAALFELD, Stephan; PREIBISCH, Stephan; TOMANCAK, Pavel. *BigDataViewer: Interactive Visualization and Image Processing for Terabyte Data Sets*. 2014. Dostupné z arXiv: 1412.0488 [q-bio.QM].

14. *Introduction to HDF5: The HDF Group* [online]. 2019-06-03 [cit. 2021-03-07]. Dostupné z: https://imagej.net/BigDataServer.

15. KOZIOL, Quincey. HDF5. In: *Encyclopedia of Parallel Computing*. Ed. PADUA, David. Boston, MA: Springer US, 2011, s. 827–833. ISBN 978-0-387-09766-4. Dostupné z DOI: 10.1007/978-0-387-09766-4_44.

16. FOLK, Mike; HEBER, Gerd; KOZIOL, Quincey; POURMAL, Elena; ROBINSON, Dana. An overview of the HDF5 technology suite and its applications. In: 2011-03, s. 36–47. Dostupné z DOI: 10.1145/1966895.1966900.

17. *HDF5 Datatypes: The HDF Group* [online]. 2009 [cit. 2021-03-07]. Dostupné z: https://support.hdfgroup.org/HDF5/doc1.6/UG/11_Datatypes.html.

18. *HDF5 File Format Specification Version 3.0: The HDF Group* [online]. 2016-04-25 [cit. 2021-03-07]. Dostupné z: https://support.hdfgroup.org/HDF5/doc/H5.format.html.

19. WILLIAMS, Lance. Pyramidal Parametrics. 1983-07, roč. 17, č. 3, s. 1–11. ISSN 0097-8930. Dostupné z DOI: 10.1145/964967.801126.

20. KRIEGER, Jan. *Spim prinziple - Wikimedia Commons: The HDF Group* [online]. 2012-03-15 [cit. 2020-12-12]. Dostupné z: https://commons.wikimedia.org/wiki/File:Spim_prinziple_en.svg.

21. BROEKE, Jurjen; PEREZ, Jose Maria Mateos; PASCAU, Javier. *Image Processing with ImageJ*. 2nd. Packt Publishing, 2015. ISBN 1785889834.

22. CHMIELEWSKI, Szymon; TOMPALSKI, Piotr. Estimating outdoor advertising media visibility with voxel-based approach. *Applied Geography*. 2017, roč. 87, s. 1–13.

23. KUBÍČEK, Milan; DUBCOVÁ, Miroslava; JANOVSKÁ, Drahoslava. *Numerické metody a algoritmy*. Vyd. 2., opr. Praha: Vysoká škola chemicko-technologická [Praha], 2005. ISBN 80-7080-558-7.

24. *Understanding Digital Image Interpolation: Cambridge in Colour* [online]. 2005 [cit. 2020-12-07]. Dostupné z: https://www.cambridgeincolour.com/tutorials/image-interpolation.htm.

25. SPRAGUE, Nathan. *Coordinate Frames: v-1.0* [online]. 2016 [cit. 2020-12-09]. Dostupné z: https://w3.cs.jmu.edu/spragunr/CS354/handouts/frames.pdf.

26. MEIJERING, Erik. *TransformJ: Affine: ImageScience Org* [online]. 1996 [cit. 2021-03-01]. Dostupné z: https://imagescience.org/meijering/software/transformj/affine/.

27. EDUARD, SOJKA; MARTIN, NĚMEC; TOMÁŠ, FABIÁN. *Matematické základy počítačové grafiky: Ostrava: VŠB* [online]. 2011 [cit. 2020-12-01]. Dostupné z: http://mi21.vsb.cz/sites/mi21.vsb.cz/files/unit/zaklady_pocitacove_grafiky.pdf.

28. HOUSE, Donald H. *Affine Transformations: In: Clemson, S.C, USA: The School of Computing at Clemson* [online] [cit. 2019-04-25]. Dostupné z: https://people.cs.clemson.edu/~dhouse/courses/401/notes/affines-matrices.pdf.

29. ŠARMAN, Arnošt; NĚMEC, Martin. *Základy počítačové grafiky*. Ostrava: VŠB, 2008. ISBN 978-80-248-1497-1.

30. NATALIA, KUKUSHKINA. *The Evolution Of React: Custom Application Development Company* [online] [cit. 2021-02-14]. Dostupné z: https://dashbouquet.com/blog/frontend-development/the-evolution-of-react.

31. PARISI, Tony. *WebGL - Up and Running: Building 3D Graphics for the Web*. O'Reilly, 2012. ISBN 978-1-4493-2357-8. Dostupné také z: http://www.oreilly.de/catalog/9781449323578/index.html.

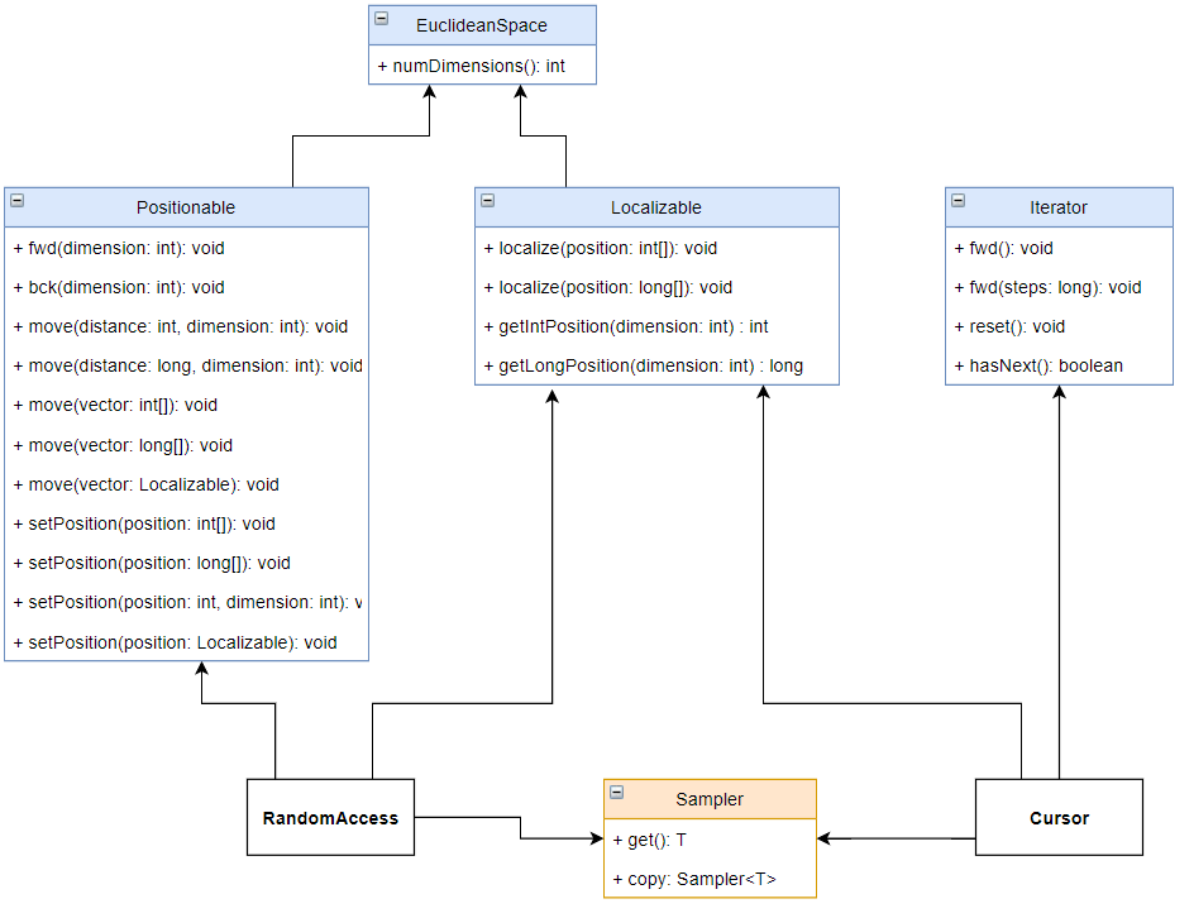
32. *WebGL Overview: The Khronos Group Inc.* [Online]. 2011-05-14 [cit. 2021-02-14]. Dostupné z: https://www.khronos.org/webgl/.

33. ROMAN, PICHLÍK. *Spring Framework – představení J2EE lightweight kontejneru* [online]. 2005-10-21 [cit. 2021-02-22]. Dostupné z: `https://www.interval.cz/clanky/spring-framework-predstaveni-j2ee-lightweight-kontejneru/`.

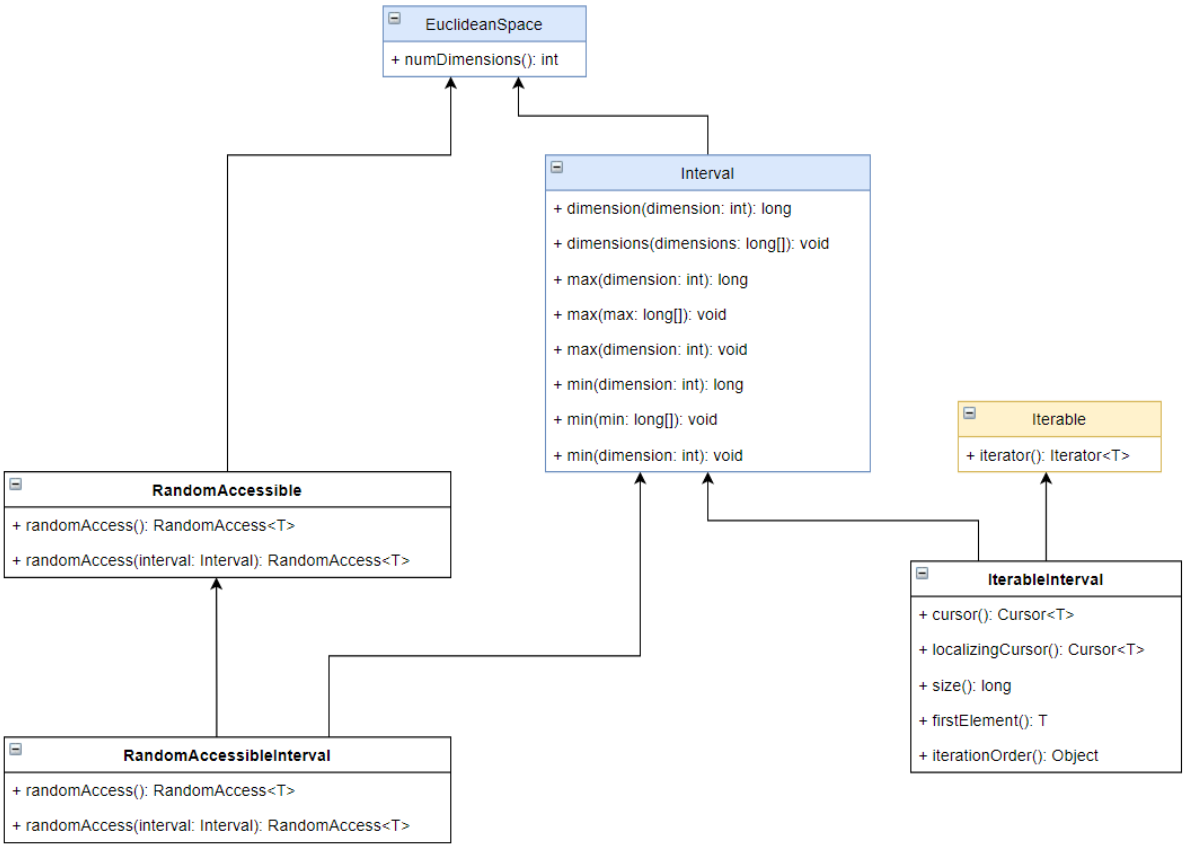
34. MEIJERING, Erik. *TransformJ: An ImageJ Plugin Suite for Geometrical Image Transformation: ImageScience Org* [online]. 1996 [cit. 2021-02-25]. Dostupné z: `https://imagescience.org/meijering/software/transformj/`.

Příloha A

ImgLib2 UML



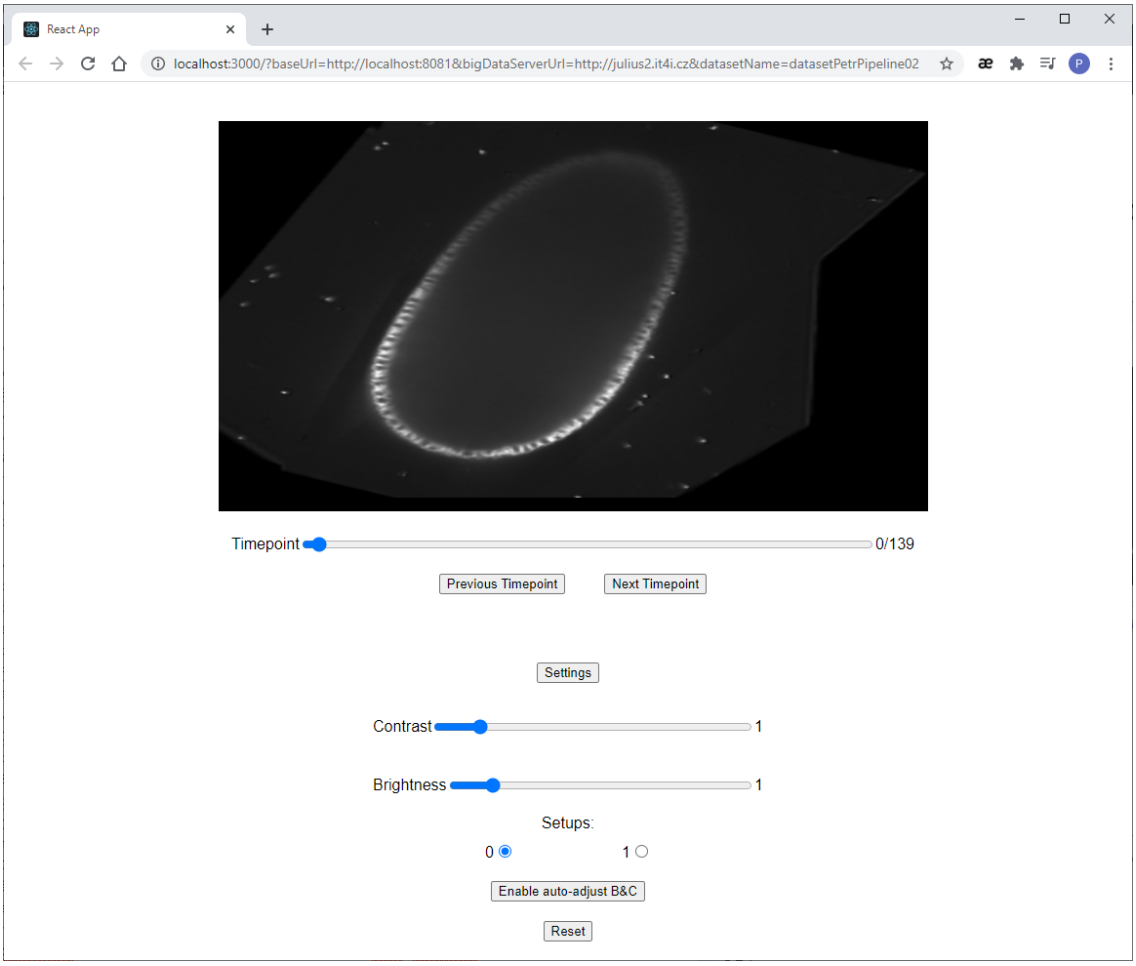
Obrázek A.1: ImgLib2 Accessors rozhraní - zjednodušený náhled



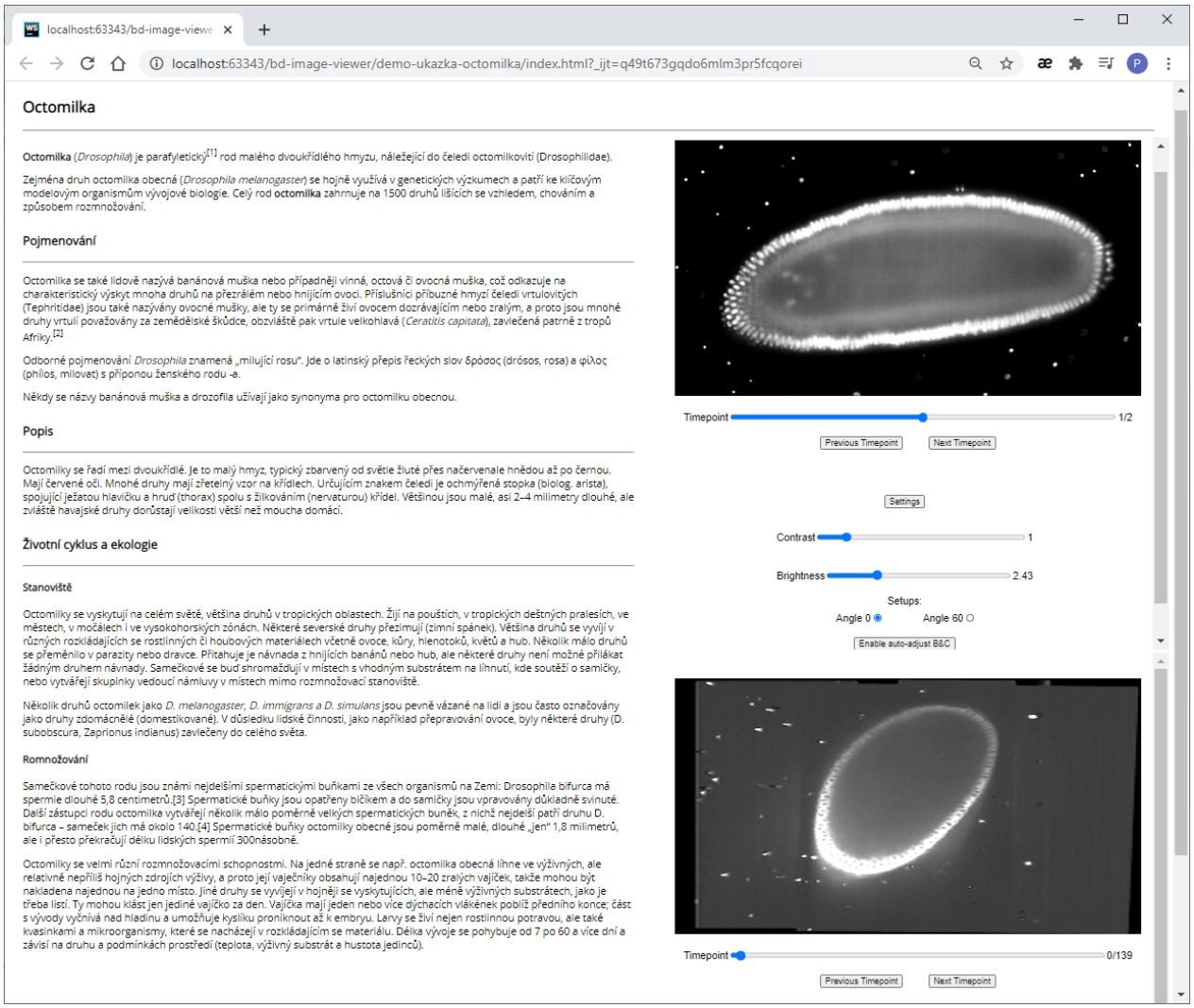
Obrázek A.2: ImgLib2 Accessible rozhraní - zjednodušený náhled

Příloha B

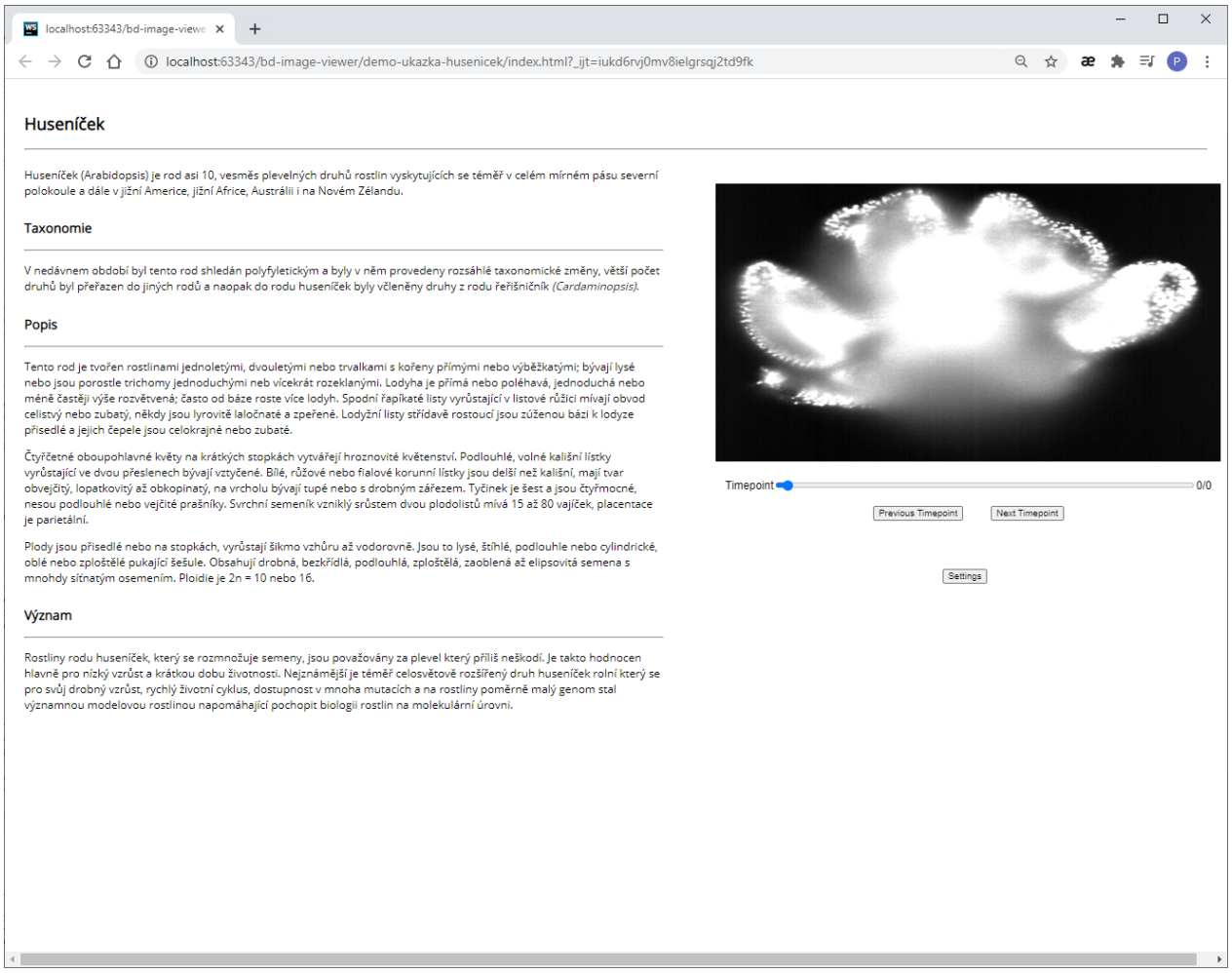
Ukázky aplikace (bd-image-viewer)



Obrázek B.1: Aplikace jako samostatná stránka



Obrázek B.2: Aplikace použita jako plugin na stránce - příklad 1



Obrázek B.3: Aplikace použita jako plugin na stránce - příklad 2